(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 472 433 A2**

# (12) EUROPEAN PATENT APPLICATION

(21) Application number : **91307748.3**

(51) Int. Cl.$^5$ : **G06F 9/26**

(22) Date of filing : **22.08.91**

(30) Priority : **23.08.90 JP 221795/90**

(43) Date of publication of application : **26.02.92 Bulletin 92/09**

(84) Designated Contracting States : **DE FR GB**

(71) Applicant : **FUJITSU LIMITED
1015, Kamikodanaka Nakahara-ku
Kawasaki-shi Kanagawa 211 (JP)**

(72) Inventor : **Maebayashi, Masato
Berubia Tsunashima Dai-4 204 2486, Shinyoshida-cho
Kohoku-ku, Yokohama-shi, Kanagawa 223 (JP)**
Inventor : **Kimura, Makoto
5-8-14, Kobayashikita, Inzaimachi
Inba-gun, Chiba 270-13 (JP)**

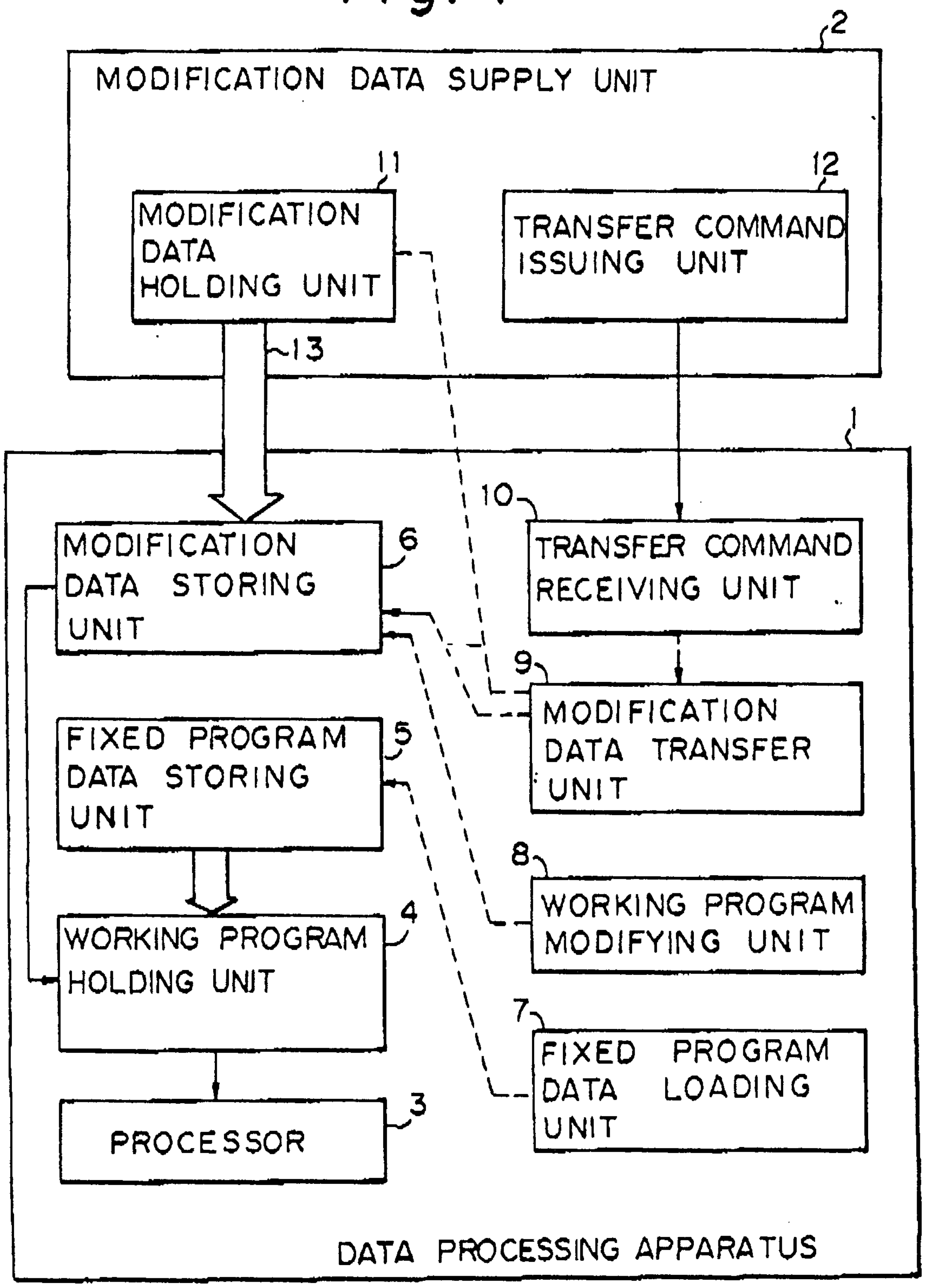
(74) Representative : **Stebbing, Timothy Charles et al
Haseltine Lake & Co. Hazlitt House 28
Southampton Buildings Chancery Lane
London WC2A 1AT (GB)**

## (54) Firmware modification system wherein older version can be retrieved.

(57) A firmware modification system in a data processing apparatus (1), containing : a processor (3) for executing a program ; a working program holding unit (4) for holding therein data of the program executed by the processor (3) ; a fixed program data storing unit (5) for storing a fixed program data ; a modification data storing unit (6) for storing modification data with which the fixed program data is to be modified, where the modification data storing unit (6) is accessible from outside of the data processing apparatus (1) for writing the modification data therein ; a fixed program data loading unit (7) for reading from the fixed program data storing unit (5), and loading the fixed program data in the working program holding unit (4) to hold the fixed program data therein ; and a working program modifying unit (8) for modifying data of the program which is held in the working program holding unit (4), with the modification data which is stored in the modification data storing unit (6).

EP 0 472 433 A2

Fig. 1



2
MODIFICATION DATA SUPPLY UNIT
11
MODIFICATION DATA HOLDING UNIT
12
TRANSFER COMMAND ISSUING UNIT
13
1
10
TRANSFER COMMAND RECEIVING UNIT
MODIFICATION DATA STORING UNIT
6
9
MODIFICATION DATA TRANSFER UNIT
FIXED PROGRAM DATA STORING UNIT
5
8
WORKING PROGRAM MODIFYING UNIT
WORKING PROGRAM HOLDING UNIT
4
7
FIXED PROGRAM DATA LOADING UNIT
PROCESSOR
3
DATA PROCESSING APPARATUS

## BACKGROUND OF THE INVENTION

### (1) Field of the Invention

The present invention relates to a firmware modification system in a data processing apparatus. The functions of one or more piece of equipment in a data processing system are often realized by firmware, i.e., the equipment contains a microprocessor and a read-only-memory (ROM), and software which determines fixed functions of the equipment is written in advance in the ROM.

### (2) Description of the Related Art

Since firmware is a kind of software, a modification thereof for upgrading the firmware or correcting a bug therein may be required. In large scale and general purpose data processing systems, a service processor and a magnetic disc device are provided therein, and software is read by the service processor from the magnetic disc device, and is installed in random access memories (RAM) in one or more piece of equipment in the data processing system when the power of the system is turned on or the system is restarted. Therefore, the service processor manages and controls the modification or correction of the firmware.

In relatively small scale data processing systems, or in a small size piece of equipment such as a peripheral interface adapter, however, firmware is written in advance in a ROM, and therefore, the ROM must be replaced with a new one when the firmware is to be modified or a bug corrected.

The system must be stopped when the ROM is replaced, and further, the work for the replacement is very bothersome, particularly when firmware in a large number of pieces of equipment is to be modified, or a plurality of pieces of equipment is distributed in a large or remote area.

Further, the data processing apparatus may not successfully operate after the modification of the program data, due to a bug included in the modified data, and in the conventional construction using the ROM only, the ROM must be again replaced in this case.

## SUMMARY OF THE INVENTION

An object of the present invention is to provide a firmware modification system of a data processing apparatus wherein a modification of firmware can be carried out fast and easily even when the data processing apparatus is located in a remote place, or when the modification must be carried out in a large number of data processing apparatuses, and program data of an older version can be retrieved when the modified program data is not successfully executed.

According to the present invention, there is provided a firmware modification system in a data processing apparatus, comprising: a processor for executing a program; a working program holding unit for holding therein data of the program executed by the processor; a fixed program data storing unit for storing a fixed program data; a modification data storing unit for storing modification data with which the fixed program data is to be modified, where the modification data storing unit is accessible from outside of the data processing apparatus for writing the modification data; a fixed program data loading unit for reading the fixed program data from the fixed program data storing unit, and loading the fixed program data in the working program holding unit to hold the fixed program data therein; and a working program modifying unit for modifying the data of the program held in the working program holding unit, with the modification data stored in the modification data storing unit.

In the above construction of the present invention, the fixed program may include program portions respectively corresponding to the functions of the fixed program data loading unit and the working program modifying unit, and the fixed program data loading unit and the working program modifying unit may be respectively realized by executions of the program portions by the processor.

In the above construction of the present invention, the firmware modification system may further comprise a modification data supply unit, provided outside of the data processing apparatus, for supplying the modification data to be stored in the modification data storing unit.

In the above construction of the present invention, the modification data supply unit may comprise: a modification data holding unit for holding the modification data to be stored in the modification data storing unit; and a transfer command issuing unit for issuing a transfer command to the data processing apparatus; the data processing apparatus further comprises a transfer command receiving unit for receiving the transfer command; the firmware modification system further comprises a modification data transfer unit for reading the modification data from the modification data holding unit, transferring the modification data from the modification data holding unit to the modification data storing unit, and writing the modification data in the modification data storing unit, when the transfer command is received by the transfer command receiving unit.

In the above construction of the present invention, the modification data may contain a plurality of versions of modification data; the firmware modification system may further comprise a modification version command unit for commanding the working program modifying unit to modify the data of the program held in the working program holding unit with the modification data up to a specific version; and the working program modifying unit may modify the data of the program held in the working program holding unit with the modification data up to the version, in the order of the versions from the oldest to the newest, when receiving the commanding of the version.

In a preferred construction of the present invention, the plurality of versions of modification data are stored in the modification data storing unit in a plurality of blocks respectively corresponding to the versions, the blocks are arrayed in the order of the versions from the oldest to the newest, and the modification version command unit commands the version by a number of blocks containing the modification data up to the version.

In a further construction of the present invention, each block for each version contains: an address of each data to be modified; new data of the address for the version; and old data of the address for a version older than the version of the block by one version level; the working program modifying unit modifies the data in the working program holding unit in the order of the versions from the oldest to the newest; and the working program modifying unit further comprising: a data comparing unit for comparing, before modifying the data in the working program holding unit, the old data of each address and the data of the same address in the working program holding unit; and an abnormal stop unit for stopping the modifying operation when the old data of each address and the data of the same address in the working program holding unit are determined by the data comparing unit to be not equal.

In a further preferred construction of the present invention, each block of the modification data contains first information indicating a version of the program data up to which version the program data held in the working program holding unit is modified with modification data of the block; the working program holding unit contains an area holding second information indicating a version of program data which is currently held therein; the working program modifying unit renews the second information indicating the version of program data, based on the first information when the program data held in the working program holding unit is modified with the modification data in each block.

In the above construction of the present invention, the working program modifying unit may comprise a version confirming unit for determining whether or not the version of the program data in the working program holding unit corresponds to a version up to which version the program data held in the working program holding unit is modified with modification data of each block, based on the first and second information, before the program data held in the working program holding unit is modified with the modification data in each block.

## BRIEF DESCRIPTION OF THE DRAWINGS

In the drawings:

Figure 1 is a diagram of the basic construction of the present invention;

Figure 2 is a diagram showing an example construction of a distributed data processing system where the present invention can be applied to the communication control processors;

Figure 3 is a diagram showing the construction of a communication control processor according to the embodiment of an present invention;

Figure 4 is a diagram showing the construction of the adapter in the communication control processor according to the embodiment of the present invention;

Figure 5 is a flowchart indicating an operation by the processor for transferring modification data to the adapter;

Figure 6 is a diagram showing an example format of the modification data stored in the EEPROM in the construction of Fig. 4;

Figure 7 is a diagram showing an example format of the header area in the modification data stored in the EEPROM in the construction of Fig. 4;

Figures 8A and 8B are diagrams indicating two types of each unit of the modification data in the modification data area in the modification data stored in the EEPROM in the construction of Fig. 4;

Figure 9 is a diagram showing an example of the end mark;

Figure 10A is a diagram showing a general format of the I/O command;

Figure 10B is a diagram showing an example format of the adapter operation descriptor AOPD;

Figure 11 is a diagram showing typical examples of the function code;

Figure 12 is a diagram showing an example format of the modification data storage area in a RAM in each of the processors in the communication control processor of Fig. 4;

Figures 13A, 13B, and 13C indicate a flowchart of a detailed operation by the processor for transferring modification data to the adapter;

Figures 14A, 14B, 14C, and 14D indicate a flowchart of a detailed operation by the adapter for transferring modification data thereto;

Figure 15 is a flowchart indicating an operation of the initial program loading in the adapter; and

Figures 16A, B, and C indicate a flowchart of a detailed operation by the adapter for loading the modification data in the RAM therein.

## DESCRIPTION OF THE PREFERRED EMBODIMENTS

### Basic Operations

Before describing the preferred embodiment of the present invention, the basic operations of the present invention are first explained below.

Figure 1 is a diagram of the basic construction of the present invention. In Fig. 1, reference numeral 1 denotes a data processing apparatus, 2 denotes a modification data supply unit, 3 denotes a processor, 4 denotes a working program holding unit, 5 denotes a fixed program data storing unit, 7 denotes a fixed program data loading unit, 8 denotes a working program modifying unit, 9 denotes a modification data transfer unit, 10 denotes a transfer command receiving unit, 11 denotes a modification data holding unit, 12 denotes a transfer command issuing unit, and 13 denotes a modification data transfer path.

Among these elements, the processor 3, the working program holding unit 4, the fixed program data storing unit 5, the modification data storing unit 6, the fixed program data loading unit 7, and the working program modifying unit 8 in the data processing apparatus 1 are indispensable constituents of the broadest scope of the present invention.

According to the above broadest scope of the present invention, the fixed program data (version 0 data) is stored in the fixed program data storing unit, and the modification data with which the fixed data is to be modified is stored in the modification data storing unit, where the modification data storing unit is accessible from outside of the data processing apparatus for writing the modification data therein. The fixed program data is read from the fixed program data storing unit by the fixed program data loading unit, and is loaded in the working program holding unit to be held therein, and then the data of the program held in the working program holding unit, is modified with the modification data stored in the modification data storing unit, by the working program modifying unit. Namely, the fixed program data (version 0 data) is permanently retained in the fixed program data storing unit. Therefore, even when it turns out that the modified program data cannot be successfully executed, the fixed program data can be retrieved at the working program holding unit. In addition, since the modification data storing unit is writable from outside of the data processing apparatus, program data of an older version can be retrieved when the modified program data is not successfully executed.

In the above construction wherein each block for each version contains: an address of each data to be modified; new data of the address for the version; and old data of the address for a version older than the version of the block by one version level, the working program modifying unit modifies the data in the working program holding unit in the order of the versions from the oldest to the newest, the data comparing unit in the working program modifying unit compares, before modifying the data in the working program holding unit, the old data of each address and the data of the same address in the working program holding unit. The modifying operation is stopped by the abnormal stop unit when the old data of each address and the data of the same address in the working program holding unit are determined by the data comparing unit to be not equal.

The remaining elements in Fig. 1 other than the above indispensable constituents operate respectively as described in the "SUMMARY OF THE INVENTION".

### Fig. 2 (Example System to which Present Invention is Applied)

Figure 2 is a diagram showing an example construction of a distributed data processing system wherein the present invention can be applied to the communication control processors 19 and 29. The distributed data processing system of Fig. 2 may be a data processing system used by a bank, a credit corporation, an insurance company, or a supermarket chain, each having a plurality of branches distributed over a large area.

In Fig. 2, reference numeral 17 and 27 each denote a host computer, 18 and 28 each denote a data base, 19 and 29 each denote a communication control processor, 20 and 30 each denote an auxiliary storage device, 21, 22, and 23 each denote a network processor, 24 denotes a maintenance center, 25 denotes a piece of terminal equipment, 26 denotes an exchange system, and 27, 28, and 29 each denote a transmission line.

The network processors 21 to 23 and the transmission lines 27 to 29 constitute a backbone network, and the network processors are located in respective nodes of the network. Although only three network processors 21 to 23 and the three transmission lines 27 to 29 are shown in Fig. 2, usually, the backbone network can be constructed in an arbitrary form. In addition, the above nodes may be located at the above branches.

The host computer 17, the data base 18, the communication control processor 19, and the auxiliary storage device 20 may constitute an information center, for example, which controle information on customers in an area, and the host computer 27, the data base 28, the communication control processor 29, and the auxiliary storage device 30 may also constitute another information center which controls information on customers in another area. The communication control processor 19 or 29 controls communications of data between the corresponding host computer and the backbone network. As well known, the function of the communication control processor includes: transformation of data format; establishing and release of data links; monitoring transmission lines; error detection; data buffering; data transfer control between the host computer and communication control processor; and the like.

The maintenance center 24 is connected to one of the network processors 21 to 23, and the exchange system 26 is connected to that network processor 22 in the construction of Fig. 2. Alternately, the exchange 26 may be connected to the other network processor when the network processor is located at the exchange. The terminal equipment 25 may be provided, for example, at each branch office of the bank, and connected to the exchange 26 through a transmission line, whereby, for example, an operator in a branch office can information from the information center through the terminal equipment, the exchange 26, and the backbone network.

## Fig. 3 (Communication Control Processor)

Figure 3 is a diagram of the construction of a communication control processor according to the embodiment of the present invention. In Fig. 3, reference numerals $31_1$ and $31_2$ each denote a common storage unit, $33_1$, $33_2$, $38_1$, and $38_2$ each denote a bus handler, $34_1$ and $34_2$ each denote an I/O device, for example, an auxiliary storage device, 35 denotes a line switching circuit, $36_1$, $36_2$, $36_3$, and $36_4$ each denote an adapter, 37 denotes a transmission line, $40_1$, $40_2$, and $40_3$ each denote a processor, $41_1$ and $41_2$ each denote a system bus, and $42_1$ and $42_2$ each denote an I/O bus.

The communication control processor of Fig. 3 is realized by a loosely-coupled multi-processor construction wherein the above-mentioned functions are shared by the processors $40_1$, $40_2$, and $40_3$. The common storage units $31_1$ and $31_2$ are used for storing data commonly used by the processors $40_1$, $40_2$, and $40_3$. The respective paired provisions of common storage units $31_1$ and $31_2$, the bus handlers $33_1$ and $33_2$, the system buses $41_1$ and $41_2$, the bus handlers $38_1$ and $38_2$, the I/O buses $42_1$ and $41_2$, the adapters $36_1$ and $36_2$, and the adapters $36_3$ and $36_4$ are made for realizing a doubled construction. Either of each pair can be used (activated) under the control of the processors $40_1$, $40_2$, and $40_3$. For example, the line switching circuit 37 is controlled to connect the transmission line 37 with the activated one of the adapters $36_3$ and $36_4$. The transmission line 37 is, for example, connected to the network processor 21 of Fig. 2. The adapters $36_3$ and $36_4$ each operate as a line interface circuit between the processors $40_1$, $40_2$, and $40_3$ and the transmission line 37, and the adapters $36_1$ and $36_2$ each operate as an I/O interface circuit between the processors $40_1$, $40_2$, and $40_3$ and the I/O devices $34_1$ and $34_2$. In this embodiment, the adapters $36_1$, $36_2$, $36_3$, and $36_4$ each correspond to the aforementioned data processing apparatus 1 in Fig. 1, and the processors $40_1$, $40_2$, and $40_3$ realize the aforementioned modification data supply unit 2 in Fig. 1.

## Fig. 4 (Adapter)

Figure 4 is a diagram showing the construction of the adapter in the communication control processor according to the embodiment of the present invention. In Fig. 4, reference numeral $42_i$ (i=1 and 2) denotes one of the above-mentioned I/O buses $42_1$ and $42_2$, $36_j$ (j=1, 2, 3, or 4) denotes one of the above-mentioned adapters $36_1$, $36_2$, $36_3$, and $36_4$, 50 denotes a bus controller, 51 denotes a microprocessor unit (MPU), 52 denotes a random access memory (RAM), 53 denotes a read only memory (ROM), 54 denotes an electrically erasable programmable ROM (EEPROM), 55 denotes a device controller, or a line interface circuit, and 56 denotes a local bus.

The MPU 51 controls the whole operation of the adapter $36_j$, and corresponds to the aforementioned processor 3 in Fig. 1. The RAM 52 is used as a main memory area (working area) of the MPU 51, and data of a program executed by the MPU 51 is loaded in the RAM 52. The RAM 52 corresponds to the aforementioned working program holding unit 4 in Fig. 1. The ROM 53 stores fixed data of the program as program data of version zero, and corresponds to the aforementioned fixed program data storing unit 5 in Fig. 1. Any of the processors $40_1$, $40_2$, and $40_3$ can write modification data in the EEPROM 54 with the aid of the MPU 51, as explained later. The modification data is used for modifying data of the program held in the RAM 52.

The program data of version zero includes a program portion for an initial program loading. The program portion is comprised of: a first routine for reading the program data of version zero from the ROM 53, and transferring the program data to the RAM 52 to load the program data in the RAM 52; and a second routine for modifying the data of the program held in the RAM 52, with the modification data held in the EEPROM 54, as explained later.

When the power of the adapter $36_j$ is turned ON, or the adapter receives a command to carry out an initial program loading operation (IPL), as explained later, the operation of the MPU 51 is jumped to a top address of an area, in the ROM 53 where the above first routine is stored, and thus the above first routine is executed by the MPU 51. Then, the operation goes to an area in the ROM 53 where the above second routine is stored. When modification data is held in the EEPROM 54, and when the modification is requested by any of the processors $40_1$, $40_2$, and $40_3$, the above second routine is executed by the MPU 51.

The device controller, or the line interface circuit 55 is provided as an interface circuit between the local bus 56 and the I/O devices $34_1$ and $34_2$, or the transmission line 37, respectively. The bus controller 50 operates as an interface circuit between the adapter $36_j$ and the I/O bus $42_i$. The functions of the bus controller 55 include: reception of a command from any one of the processors $40_1$, $40_2$, and $40_3$; transfer of the received command to the MPU 51; buffering of data to be transferred between one of the processors $40_1$, $40_2$, and $40_3$ and the I/O devices $34_1$ and $34_2$, or the transmission line 37; buffering of the modification data to be transferred from one of the processors $40_1$, $40_2$, and $40_3$ to the EEPROM 54; obtaining a right to access a memory (not shown) in any of the processors $40_1$, $40_2$, and $40_3$; transfer of the modification data from the memory in one of the processors $40_1$, $40_2$, and $40_3$ to the bus controller 50; and transfer of the modification data from the bus controller 50 to the EEPROM 54. The above transfer operations of the modification data are realized by DMA (direct memory access) operations, i.e., the bus controller 50 comprises a DMA controller. The above function of access to a memory (not shown) in any of the processors $40_1$, $40_2$, and $40_3$ is used, for example, when transferring the modification data from the memory in the processor to the bus controller 50.

## Fig. 5 (Operation of Processor)

In the distributed data processing system of Fig. 2, the modification data with which the program data of the program in the RAM 52 of Fig. 4 is to be modified, is transferred from the maintenance center 24 to one of the processors $40_1$, $40_2$, and $40_3$ in the communication control processor 19 or 29, through the backbone network. The transferred modification data is held in a memory (RAM, not shown) in the processor. When receiving the modification data, the processor carries out the operation of Fig. 5 to transfer the modification data to the adapter $36_i$. Figure 5 is a flowchart indicating an operation by the processor for transferring modification data to one of the adapters $36_1$, $36_2$, $36_3$, and $36_4$. The processor $40_1$, $40_2$, or $40_3$ in the communication control processor 19 or 29 may transfer the modification data to the adapters $36_1$, $36_2$, $36_3$ and $36_4$ only when the processor receives a request to transfer same or when a processing load on the processor becomes small after the above reception of the modification data.

In step 101, the processor issues an adapter state transition command to the adapter, and in step 102 it is determined whether or not the execution of the adapter state transition command is successfully completed. When it is determined that the operation is not successfully completed in step 102, the operation goes to step 109 to execute an abnormal end (abnormal termination) routine. When it is determined that the execution of the adapter state transition command is successfully completed in step 102, the operation goes to step 103. The above adapter state transition command is first received at the bus controller 50 in the adapter, and then, the bus controller 50 supplies an interrupt signal corresponding to the adapter state transition command to the MPU 51. In response to the interrupt signal, the MPU 51 executes a routine to bring the adapter to a state whereby the adapter is ready to receive the modification data, and returns to the processor of the communication control processor a ready signal indicating that the adapter is ready to receive the modification data. When the processor receive the ready signal in step 103, the operation goes to step 104 to issue a modification data transfer command to the adapter. The details of the operation of the transfer are explained later. In step 105, it is determined whether or not the transfer of the modification data has been successfully completed. When the transfer cannot be successfully completed, the operation goes to step 109 to execute the abnormal end routine, and when it is determined that the transfer is successfully completed, the operation goes to step 106 to issue an adapter operation restart command to the adapter. The adapter operation restart command is first received at the bus controller 50 in the adapter, and then, the bus controller 50 supplies an interrupt signal corresponding to the adapter operation restart command to the MPU 51. In response to the interrupt signal, the MPU 51 executes a routine to bring the adapter to a normal processing state, and returns to the processor of the communication control processor a complete signal indicating that the adapter is in the normal processing state. In step 107, it is determined whether or not the adapter operation restart command is successfully completed. When the above complete signal is not received by the proceseor in the communication control processor, the operation goes to step 109 to execute the abnormal end routine, and when the above complete signal is received by the processor in the communication control processor, the operation of Fig. 5 is completed.

## Figs. 6, 7, 8A, 8B, and 9 (Format of Modification Data)

Figure 6 is a diagram showing an example format of the modification data stored in the EEPROM 54 in the construction of Fig. 4. As shown at the left side of Fig. 6, the EEPROM 54 contains control data, a plurality of modification data blocks 1 to N, and a vacant area. The plurality of modification data blocks 1 to N respectively correspond to a plurality of versions of modification data, i. e., a plurality of revisions. The order of the modification data blocks corresponds to the order of the number of the revisions. That is, the modification data block 1 corresponds to the modification data for the first revision 1; the modification data block 2 corresponds to the modification data for the second revision; ··· the modification data block N corresponds to the modification data for the N-th revision. As shown at the upper right side of Fig. 6, the area of the control data contains: a fixed value "FPDT (ASCII CODE)" which indicates a top of the area of the control data; a status of the EEPROM 54 which indicates whether or not the EEPROM is protected (closed) against a writing operation ("00" indicates that the EEPROM is protected, and "01" indicates that the EEPROM is open (not protected) for a writing operation); an identification number PM-ID which indicates the processor in the communication control processor which supplies the commands to the adapter; a number N′ of modification data blocks to be used for modification; the size of the vacant area (the remaining area) of the EEPROM 54; the size of the area in which the modification data is written in the EEPROM 54; and the check sum which is a sum of all data in the control data and the modification data blocks in the EEPROM 54. As explained later, the above control data except the check sum is supplied from one of the processors $40_1$, $40_2$, and $40_3$ to the adapter.

As shown at the lower right side of Fig. 6, each modification data block contains a header area, modification data area, and an end mark.

Figure 7 is a diagram showing an example format of the header area of the modification data block of Fig. 6.

As shown in Fig. 7, the header area contains: a fixed value "DBLK (ASCII CODE)"; a current version of the program data which is held on the RAM 52 in the adapter, and which is to be further modified; a new version of the program data to which version the program data held on the RAM 52 is modified with the modification data of the modification data block; the size of the modification data block; and a check sum of the data in the modification data block.

In each of the above modification data blocks, the modification data is contained in a plurality of units. Figures 8A and 8B are diagrams indicating two types of each unit of the modification data in the modification data area in the modification data stored in the EEPROM in the construction of Fig. 4. In the type of Fig. 8A, each unit contains old data and new data for two bytes with a corresponding address of the RAM 52. On the other hand, in the type of Fig. 8B, each unit contains new data for four bytes with a corresponding address of the RAM 52. Each unit also contains information which indicates which type the above unit is.

Figure 9 is a diagram showing an example of the end mark. The end mark is a fixed value "FFFFFFFF".

## Fig. 10A (I/O Command)

Figure 10A is a diagram showing a general format of an I/O command which is issued to the adapters $36_1$, $36_2$, $36_3$, and $36_4$ from the processors $40_1$, $40_2$, and $40_3$ in the communication control processor. The format of Fig. 10A is used in all commands issued from the processors $40_1$, $40_2$, and $40_3$ toward the adapters $36_1$, $36_2$, $36_3$, and $36_4$ through the I/O buses $42_1$ and $42_2$. The format contains: a number ADP-NO. of an adapter to which the I/O command is issued; an operation code OPECD which indicates a type of the command; and an AOPD address. When the I/O command is a command which commands an operation which should be initiated by the MPU 51 in the adapter, the OPECD is indicated as "IAD". The AOPD represente an adapter operation descriptor AOPD, and the AOPD address indicates a top addrese of an area of the memory (RAM) in the processor at which the adapter operation descriptor AOPD is written.

## Fig. 10B (Adapter Operation Descriptor AOPD)

Figure 10B is a diagram showing an example format of the adapter operation descriptor AOPD. The adapter operation descriptor AOPD of Fig. 10B contains areas for: a command code CMDCODE which indicates a type of an operation requested by the command; a data count DATA COUNT which indicates a number of bytes of the modification data; a top address of modification data storage area (which is explained later); a function code FNC CODE which indicates a detail of the function of the operation requested by the command; and a number N′ of the blocks of the modification data which is to be used for the modification.

When the above AOPD is used for a transfer command of the modification data from one of the processors $40_1$, $40_2$, and $40_3$ to one of the adapters $36_1$, $36_2$, $36_3$, and $36_4$, a command code for the transfer command is set in the above area for the command code CMDCODE, the top address of the modification data storage area is set in the above area for that address, and the number N′ of the blocks of the modification data to be used for the modification is set in the above area for the number of the blocks of the modification data.

When the above AOPD is used for an initial program loading command from one of the processors $40_1$, $40_2$, and $40_3$ to the adapters $36_1$ $36_2$, $36_3$, and $36_4$, a command code for the initial program loading command is set in the above area for the command code CMDCODE, and all zero is set in the above area for that address. Another number N′ of the blocks of the modification data which is to be used in the execution of the initial program loading command, which number N′ is different from the number N′ stored in the EEPROM 54 at this time, may be set in the above area for the number N′ in the AOPD to carry out a modification based on the new number N′ in response to the initial program loading command.

Thus, the processor in the communication control processor can supply the adapter the number N′ of modification data blocks which is to be used in the modification of the program data in the RAM 52 in the adapter. Then, the modification of the program data (firmware) can be carried out when an initial program loading command is issued from the processor to the adapter.

Since each modification data block corresponds to one version of the program data, the processor can change the version of the program data in the RAM 52 in the adapter, arbitrarily within the versions zero to N, where N is the number of all modification data block stored in the EEPROM 54. For example, when the processor detects that the adapter does not operate normally with a specific version of the firmware, the processor can change the version to an older version as explained above.

## Fig. 11 (Function Code FNC CODE)

Figure 11 is a diagram showing typical examples of the function code regarding operations for writing in the EEPROM 54 in the adapters. As shown in Fig. 11, the function code FNC CODE is "00" when one of the processors $40_1$, $40_2$, and $40_3$ commands a new writing of modification data in the EEPROM 54; the function code FNC CODE is "01" or "02" when one of the processors $40_1$, $40_2$, and $40_3$ commands a writing of modification data in addition to modification data which is previously mitten in the EEPROM 54; the function code FNC CODE is "03" when the processor commands a closing of the EEPROM 54 (make the EEPROM to be protected against a writing operation); the function code FNC CODE is "04" when the processor commands a clearing and then closing of the EEPROM 54, and an initializing of the control data; and the function code FNC CODE is "05" when the processor commands a verifying of the modification data in the modification data blocks designated in the area of the control data with regard to the block format, consistency of the new and old versions, and the check sum.

## Fig. 12 (Format of Modification Data Storage Area in RAM)

Figure 12 is a diagram showing an example format of the modification data storage area which is held in a RAM in one of the processors in the communication control processor of Fig. 4. As shown in Fig. 12, the format of the modification data storage area is the same as the aforementioned format of the modification data in the EEPROM 54 shown in Fig. 6.

## Figs. 13A, 13B, and 13C (Detailed Operation by Processor for Transferring Modification Data to Adapter)

Figures 13A, 13B, and 13C indicate a flowchart of a detailed operation by the processor for transferring modification data to the adapter.

In step 301, the processor stores an AOPD in the memory which is provided therein, where the above function code FNC CODE is set as "00" or "01". Then, in step 302, the processor stores modification data in the memory. In step 303, the processor issues an IAD command (an I/O command wherein the operation code OPECD is set as "IAD"). The dashed lines and the encircled symbols "A" and "B" indicates relationships between the corresponding operations in the adapter which are indicated in Figs. 14A, 14B, and 14C.

In step 304, it is determined whether or not an end interrupt signal is received. As explained later with reference to Fig. 14D, the adapter returns an end interrupt signal to the processor when an operation for an IAD command is completed. When it is determined that the end interrupt signal is received, the operation goes to step 305 of Fig. 13B. In step 305, it is determined whether or not the processor has further (additional) modification data to be transferred to the adapter. When it is determined thai the processor has further modification data, the operation goes to step 306, and when it is determined that the processor has not additional modification data, the operation goes to step 311 of Fig. 13C.

In step 306, the processor stores in the memory therein an AOPD wherein the function code FNC CODE is set as "02", and then stores the additional modification data in the memory in step 307. Next, in step 308, the processor issued an IAD command for transferring the additional modification data to the adapter.

In step 309, it is determined whether or not an end interrupt signal is received, and when it is determined that the end interrupt signal is received, the operation goes to step 310. In step 310, it is determined whether or not the processor has further (additional) modification data to be transferred to the adapter. When it is determined that the processor has further modification data, the operation goes back to step 306, and when it is determined that the processor has no additional modification data, the operation goes to step 311 of Fig. 13C.

In step 311, the processor stores an AOPD wherein the function code FNC CODE is set as "03", and then issues an IAD command to the adapter for closing the EEPROM 54. Then, when an end interrupt signal is received in step 313, the operation of Figs. 13A to 13C is completed.

## Figs.14A, 14B, 14C, and 14D (Detailed Operation by Adapter for Transferring Modification Data)

Figures 14A, 14B, 14C, and 14D indicate a flowchart of a detailed operation by the adapter for transferring modification data thereto.

The operation of 14A, 14B, 14C, and 14D starts when an IAD command is received by the adapter from one of the processors $40_1$, $40_2$, and $40_3$ in step 400. In step 401, the MPU 51 reads out the AOPD (adapter operation descriptor) through the bus controller 50 from the memory in the processor which issued the IAD command. The AOPD address in the IAD command shown in Fig. 10A is used to the reading operation. The data of the AOPD is transferred to the adapter by using the DMA function of the bus controller 50. In step 402, the next step is determined according to the function code FNC CODE in the AOPD. When the function code FNC CODE is equal to "00", the operation goes to step 403 in Fig. 14B, when the function code FNC CODE is equal to "01" or "02", the operation goes to step 406 in Fig. 14B, and when the function code FNC CODE is equal to "03", the operation goes to step 410 in Fig. 14C.

In step 403, the MPU 51 reads the content of the modification data storage area (Fig. 12), where the top address of the modification data storage area is read from the above AOPD, and then transfers the modification data to the EEPROM 54 in the adapter. The DMA function of the bus controller 50 is used for the transfer operations between the memory in the processor and the bus controller 50, and between the bus controller 50 and the EEPROM 54.

In this embodiment, the modification data is transferred to the adapter with the aid of the MPU 51 in the adapter through the I/O bus $42_1$ or $42_2$, the bus controller 50, and the local bus 56. Namely, the modification data path 13 in Fig. 1 is realized by the I/O bus $42_1$ or 422, the bus controller 50, and the local bus 56, and the modification data transfer unit 9 in Fig. 1 is realized by the MPU 51 and the bus controller 50.

Then, the transferred data is stored in an area which follows the area of the control data as shown in Fig. 6 in the EEPROM 54 in step 404. Then, the opetation goes to step 405 to renew the control data, and store the renewed control data in the EEPROM 54. In the renewing operation, the number N′ of the modification data to be used for the modification in the AOPD is written in the area of the control data, the status is set as "open (not protected)", an identification number of the processor which issued the IAD command is set as the number PM-ID, and the sizes of the vacant area and the written area are renewed based on the transferred modification data. Then the operation goes to step 411 in Fig. 14D to send the end interrupt signal to the processor.

In step 406, the MPU 51 reads the content of the modification data storage area, where the top address of the modification data storage area is read from the above AOPD, and then transfers the modification data to the EEPROM 54 in the adapter. The DMA function of the bus controller 50 is used for the transfer operations between the memory in the processor and the bus controller 50, and between the bus controller 50 and the EEPROM 54. The transferred data is stored in an area which follows the area where the modification data is previously stored in the EEPROM 54 in step 407. Then, the operation goes to step 408 to determine whether or not the function code FNC CODE is "01" or "02". When the function code FNC CODE is determined to be "01", the operation goes to the above-explained step 405. When the function code FNC CODE is determined to be "02", the operation goes to step 409. In step 409, the MPU 51 renews the control data, and store the renewed control data in the EEPROM 54. In the renewing operation, the number N′ of the modification data to be used for the modification in the AOPD is written in the area of the control data, the sizes of the vacant area and the written area are renewed based on the transferred modification data. Then the operation goes to step 411 in Fig. 14D to send the end interrupt signal to the processor.

In step 410, the MPU 51 renews the control data, and stores the renewed control data in the EEPROM 54. In the renewing operation, the number N′ of the modification data to be used for the modification in the AOPD is written in the area of the control data, the status is set as "closed (protected)", the number of the modification data blocks is renewed, the sizes of the vacant area and the written area are renewed, and the check sum of the data in the area where the modification data is stored is calculated. Then the operation goes to step 411 in Fig. 14D, to send the end interrupt signal to the processor.

Figs. 15, 16A, 16B, 16C, and 16d (Initial Program Loading in Adapter)

When an initial program loading command is issued to an adapter by one of the processor $40_1$, $40_2$, and $40_3$ in the communication control processor, the adapter executes the process of Fig. 15. Figure 15 is a flowchart indicating an operation of the initial program loading in an adapter.

In step 201, the operation goes to a predetermined address of the ROM 53 in the adapter to read out a fixed program data (program data of version zero) from the ROM 53, and to load the program data in the RAM 52 in the adapter. Then, in step 202, the MPU 51 reads the number N′ of the modification data blocks to be used, from the adapter operation descriptor AOPD to determine whether or not modification data stored in the EEPROM 54 in step 203. When the number N′ is determined to be zero, the operation of Fig. 15 (the initial program loading) is completed. When the number N′ is determined not to be zero, the operation goes to step 204 to carry out the operation of Figs. 16A to 16C for modifying the program data held in the RAM 52. When the operation of Figs. 16A to 16C is completed, the operation of Fig. 15 is completed.

Figures 16A, B, and C indicate a flowchart of a detailed operation by the adapter for loading the modification data in the RAM therein.

In step 501 of Fig. 16A, it is determined whether or not the check sum in the control data area is correct. When it is determined that the check sum is not correct, the operation of Figs. 16A to 16C is completed. When it is determined that the check sum is correct, the operation goes to step 502. In step 502, the MPU 51 holds at a predetermined register therein a number N′ of modification data blocks, where the number N′ is a maximum number of modification data blocks which are to be used in the operation of modifying the program data in the RAM 52. Namely, the modification is to be carried out by using the modification data corresponding to the block numbers from 1 to the number N′. This number N′ can be supplied from the processor in the communication control processor to the adapter by issuing an IAD command such as a transfer command for the modification data, and an initial program loading command using the formats using the command formats of Figs. 10A and 10B, as explained before with reference to Fig. 10B. The above number N′ can be contained in the format of the AOPD of Fig. 10B.

Next, in step 503, the MPU 51 sets an initial value "1" as a variable M. Then, the MPU 51 reads a modification data block whose block number is equal to the above variable M from the EEPROM 54, and then it is determined whether or not the check sum in the header area (Fig. 7) in each modification data block is correct, in step 505. When it is determined that the check sum is not correct in step 505, the operation of Figs. 16A to 16C is completed. When it is determined that the check sum is correct in step 505, the operation goes to step 506 to determine whether or not the information on the "current version" in the header area (Fig. 7) is equal to an indication of "current version" in the content of the RAM 52 in the adapter. As explained later, when the program data held on the RAM 52 is modified with the modification data of each modification data block, a number indicating the new current version is written in a predetermined area of the RAM 52. When it is determined not equal in step 506, the operation of Figs. 16A to 16C ends (abnormal end). When it is determined equal in step 506, the operation goes to step 507 to read out modification data in the top address (of the first unit) of the modification data block.

In step 508, the above-mentioned type of the modification data of the unit is determined. When the type is determined to be "00" the operation goes to step 509. When the type is determined to be "01" the operation goes to step 511. In step 509, the MPU 51 reads out the program data in the address of the above unit from the RAM 52, as a current data, and it is determined whether or not the current data is equal to the old data (Fig. 8A) in the unit in step 510. When it is determined to be not equal in step 510, the operation of Figs. 16A to 16C is completed. When it is determined equal in step 510, the operation goes to step 511.

In step 511, the MPU 51 reads out modification data in the next unit. Then, in step 512, it is determined whether or not the end mark is detected. When the end mark is detected, the operation goes to step 513. When the end mark is not detected, the operation goes back to step 508 for the operation of the next unit.

In step 513, the MPU 51 reads out modification data in the top address (of the first unit) of the modification data block again from the EEPROM 54. Then, in step 514, the MPU 51 rewrites, in the RAM 52, the program data of the same address as that in the unit, with the modification data of the unit. Then, in step 515, the MPU 51 reads out modification data in the next unit. In step 516, it is determined whether or not the end mark is detected. When the end mark is detected, the operation goes to step 517. When the end mark is not detected, the operation goes back to step 514.

In step 517, the MPU 51 rewrites the information on the above-mentioned number indicating the new current version. Then, in step 518, the above-mentioned variable M is incremented by one. In step 519, it is determined whether or not the above variable M reaches the above number N′. When it is determined that the variable M reaches the number N′, the operation of Figs. 16A to 16C is completed. When it is determined that the variable M does not reach the number N′, the operation goes back to step 504 to carry out the modification with the modification data of the next new version.

## Other Matters

As explained above, according to the above construction, a modification of firmware can be carried out fast and easily. In addition, for example, as shown in the system of Fig. 2, the transfer of modification data to a remote data processing apparatus and a command to modify program data in a remote data processing apparatus, are possible through the network. Further, it is easy for the processors in the communication control processor to command a number of data processing apparatuses to modify program data in their apparatuses. Namely, the modification can be carried out in a large number of data processing apparatuses.

Another advantage is that program data of an older version can be retrieved when it turns out that modified program data is not successfully executed. For example, in the above embodiment, when one of the processors $40_1$, $40_2$, and $40_3$ in the communication control processor detects a malfunction of the adapters after a modification of the program data up to a first version number, the processor can make the adapter retrieve a second version which is older than the above first version. This is carried out by issuing an IAD command such as the transfer command for the modification data and the initial program loading command using the formats of Figs. 10A and 10B to the adapters wherein the number N′ of modification data blocks which corresponds to the above older version is set in the aforementioned area for the number N′ of modification data blocks in the AOPD of Fig. 10B. Namely, the above number N′ is a number of the modification data blocks to be used for modification by which the version of the program data in the RAM 52 is made the above second version. As explained with reference to Figs. 15 to 16D, when the adapter receives the above initial program loading command, the operations of Figs. 15 to 16D are carried out, and therefore, the version of the program data in the RAM 52 is made the above second version.

Although, in the above embodiment, the modification data path 13 in Fig. 1 is realized by the I/O bus $42_1$ or 422, the bus controller 50, and the local bus 56, and the modification data transfer unit 9 in Fig. 1 is realized by the MPU 51 and the bus controller 50, the modification data path 13 may be provided, in addition to the local path 50 of Fig. 4, for the processor to directly access the EEPROM 54.

## Claims

1. A firmware modification system in a data processing apparatus (1), comprising:

a processor (3) for executing a program;

a working grogram holding means (4) for holding therein data of said program executed by said processor (3);

a fixed program data storing means (5) for storing a fixed program data;

a modification data storing means (6) for storing modification data with which said fixed program data is to be modified, where said modification data storing means (6) is accessible from outside of said data processing apparatus (1) for writing the modification data therein;

a fixed program data loading means (7) for reading said fixed program data from said fixed program data storing means (5), and loading the fixed program data in said working program holding means (4) to hold said fixed program data therein; and

a working program modifying means (8) for modifying said data of the program held in said working program holding means (4), with said modification data stored in said modification data storing means (6).

2. A firmware modification system according to claim 1, wherein said fixed program data includes program portions respectively corresponding to the functions of said fixed program data loading means (7) and said working program modifying means (8), and said fixed program data loading means (7) and said working program modifying means (8) are respectively realized by executions of said program portions by said processor (3).

3. A firmware modification system according to claim 1, further comprising:

a modification data supply means (2), provided outside of said data processing apparatus (1), for supplying the modification data which is to be stored in said modification data storing means (6).

4. A firmware modification system according to claim 3, wherein said modification data supply means (2) comprises:

a modification data holding means (11) for holding said modification data which is to be stored in said modification data storing means (6); and

a transfer command issuing means (12) for issuing a transfer command to said data processing apparatus (1);

said firmware modification system further comprises a modification data transfer path (13) from said modification data holding means (11) to said modification data storing means;

said data processing apparatus (1) further comprises a transfer command receiving means for receiving said transfer command;

said firmware modification system further comprises a modification data transfer means (10) for reading said modification data from said modification data holding means (11), transferring the modification data from the modification data holding means (11) through said modification data transfer path (13) to said modification data storing means (6), and writing the modification data in the modification data storing means (6), when said transfer command is received by said transfer command receiving means.

5. A firmware modification system according to claim 1, wherein said modification data contains a plurality of versions of modification data;

said firmware modification system further comprises a modification version command means for commanding said working program modifying means (8) to modify the data of the program held in said working program holding means (4) with the modification data up to a specific version; and

said working program modifying means (8) modifies the data of the program held in said working program holding means (4) with the modification data up to said version in the order of the versions from the oldest to the newest when receiving said commanding of the version.

6. A firmware modification system according to claim 5, wherein said plurality of versions of modification data are stored in said modification data storing means (6) in a plurality of blocks respectively corresponding to the versions, and the blocks are arrayed in the order of the versions from the oldest to the newest; and

   said modification version command means commands said version by a number of blocks containing the modification data up to said specific version.

7. A firmware modification system according to claim 1, wherein each block for each version contains:

   an address of each data which is to be modified; new data of the address for the version; and old data of the address for a version which is older than the version of the block by one version level;

   said working program modifying means (8) modifies the data in said working program holding means (4) in the order of the versions from the oldest to the newest; and

   said working program modifying means (8) further comprising:

   a data comparing means for comparing, before modifying the data in said working program holding means (4), said old data of each address and the data of the same address in said working program holding means (4); and

   an abnormal stop means for stopping the modifying operation when said old data of each address and the data of the same address in said working program holding means (4) are determined to be not equal by said data comparing means.

8. A firmware modification system according to claim 6, wherein each block of the modification data contains first information indicating a version of the program data up to which version the program data held in said working program holding means is modified with modification data of the block;

   said working program holding means (4) contains an area holding second information indicating a version of program data which is currently held therein;

   said working grogram modifying means (8) renews said second information indicating the version of program data, based on said first information when the program data held in said working program holding means is modified with the modification data in each block.

9. A firmware modification system according to claim 6, wherein said working program modifying means (8) comprises a version confirming means for determing whether or not the version of the progrm data in said working progrm holding means corresponds to a version up to which version the program data held in the working program holding means is modified with modification data of each block, based on said first and second information, before the program data held in the working program holding means is modified with the modification data in each block.

10. A firmware modification system according to claim 1, wherein said fixed program data storing means (5) is realized by a read only memory.

Fig. 1

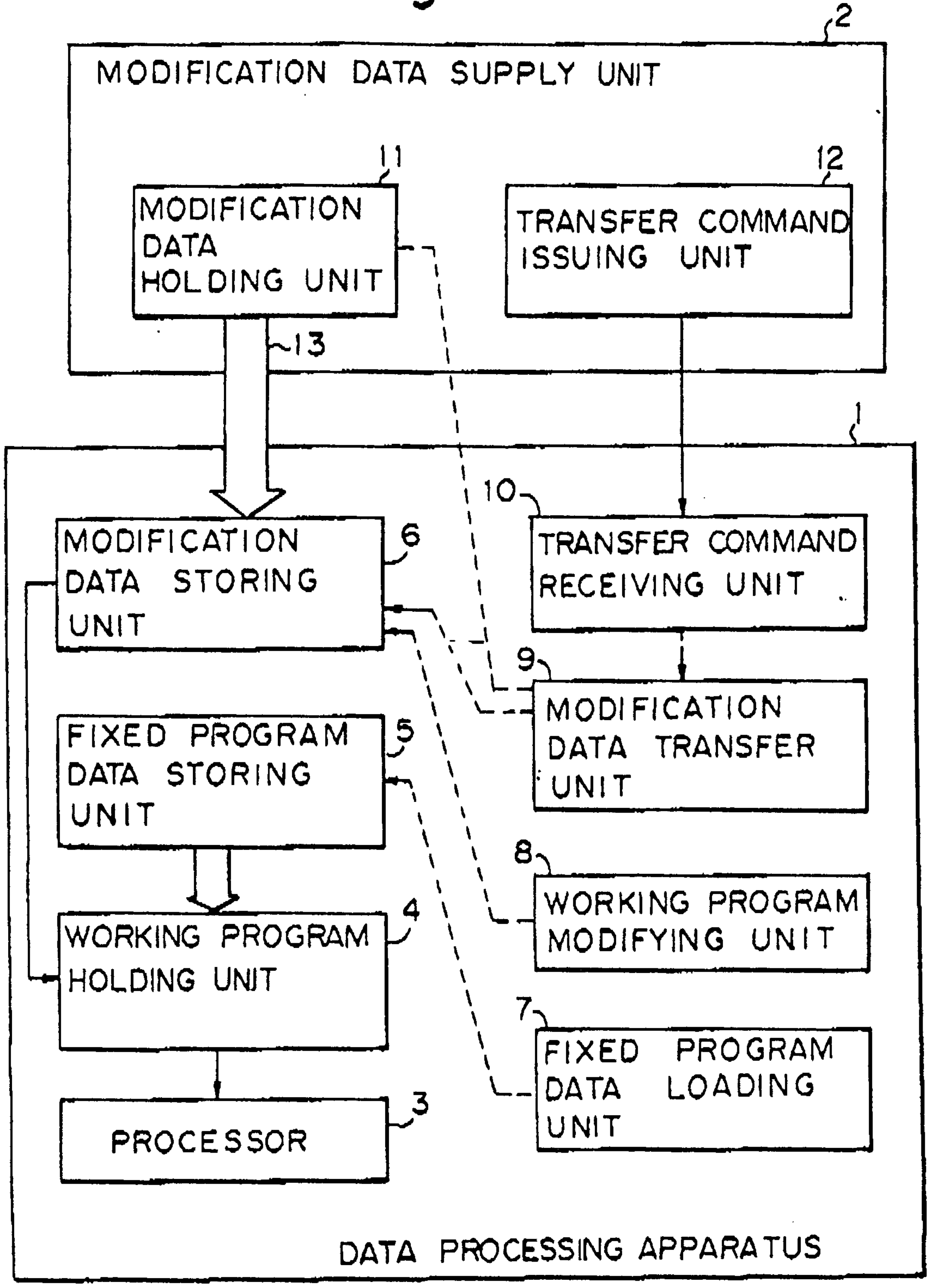

2
MODIFICATION DATA SUPPLY UNIT
11
MODIFICATION DATA HOLDING UNIT
12
TRANSFER COMMAND ISSUING UNIT
13
1
10
TRANSFER COMMAND RECEIVING UNIT
MODIFICATION DATA STORING UNIT
6
9
MODIFICATION DATA TRANSFER UNIT
FIXED PROGRAM DATA STORING UNIT
5
8
WORKING PROGRAM MODIFYING UNIT
WORKING PROGRAM HOLDING UNIT
4
7
FIXED PROGRAM DATA LOADING UNIT
PROCESSOR
3
DATA PROCESSING APPARATUS

Fig. 2

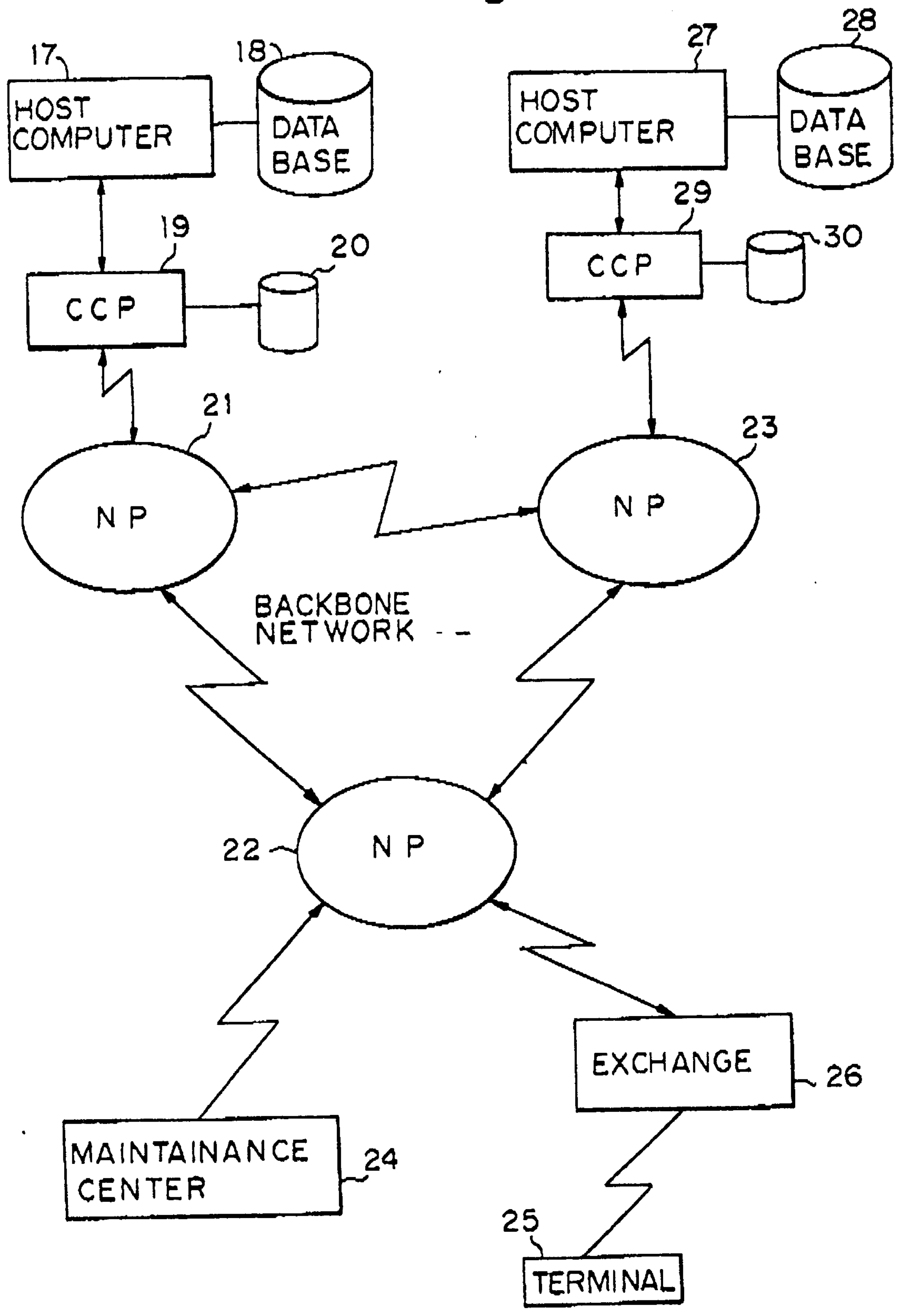
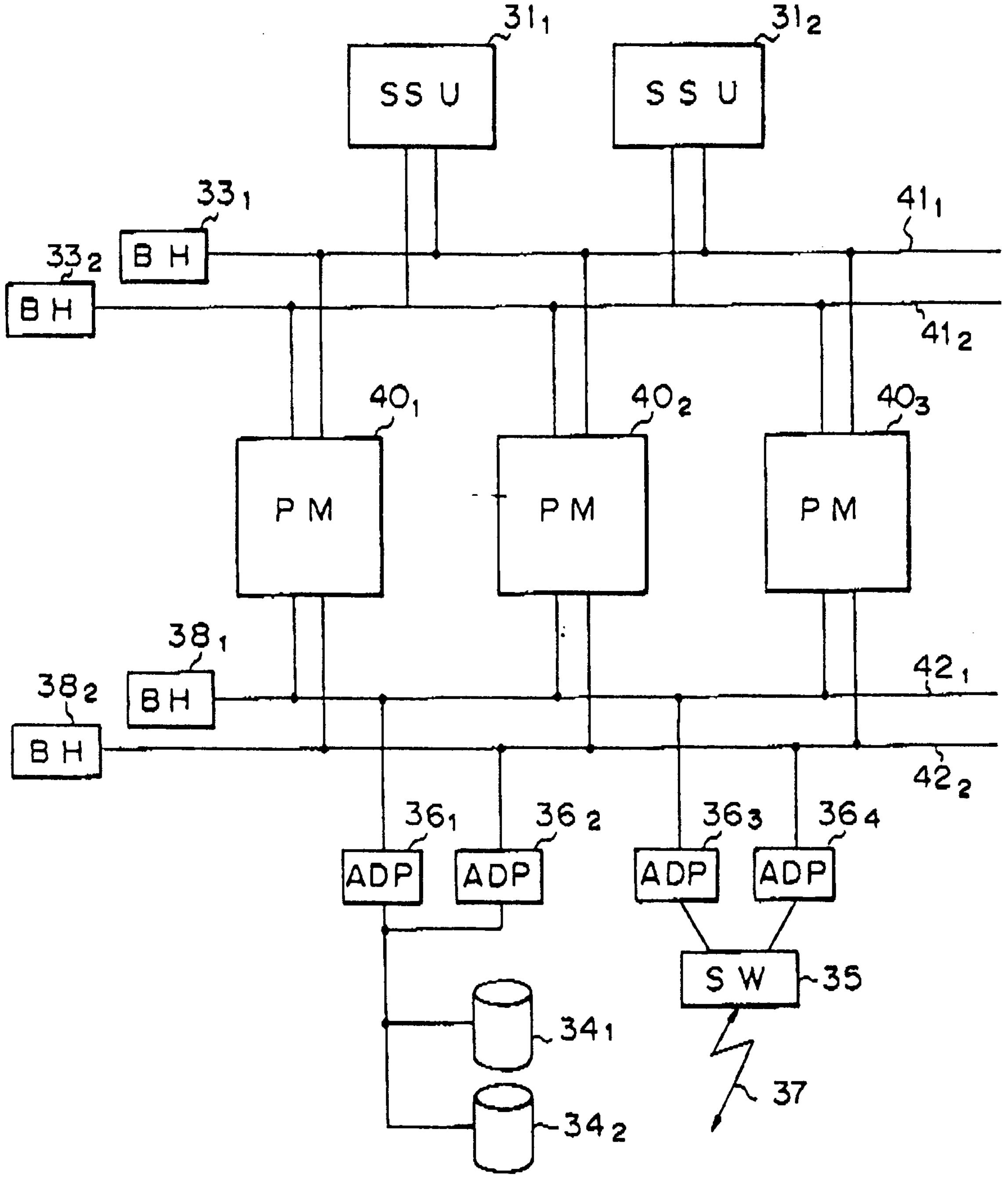

17
HOST COMPUTER
18
DATA BASE
19
CCP
20
27
HOST COMPUTER
28
DATA BASE
29
CCP
30
21
N P
23
N P
BACKBONE NETWORK
22
N P
MAINTAINANCE CENTER
24
EXCHANGE
26
25
TERMINAL

Fig. 3



SSU
$31_1$
SSU
$31_2$
BH
$33_1$
BH
$33_2$
$41_1$
$41_2$
PM
$40_1$
PM
$40_2$
PM
$40_3$
BH
$38_1$
BH
$38_2$
$42_1$
$42_2$
ADP
$36_1$
ADP
$36_2$
ADP
$36_3$
ADP
$36_4$
SW
35
$34_1$
$34_2$
37

Fig. 4

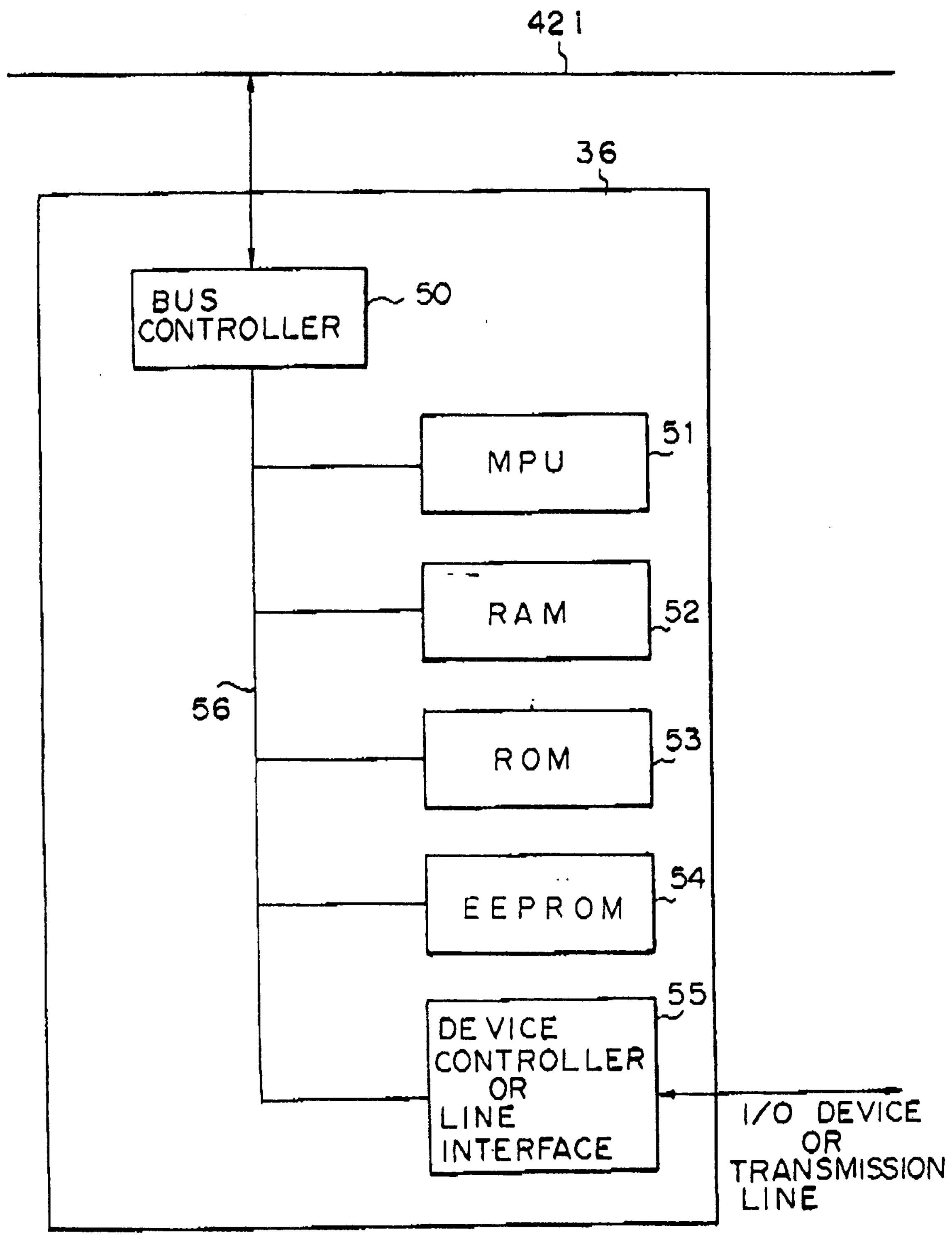

421
36
BUS CONTROLLER
50
MPU
51
RAM
52
56
ROM
53
EEPROM
54
55
DEVICE CONTROLLER OR LINE INTERFACE
I/O DEVICE OR TRANSMISSION LINE

Fig. 5

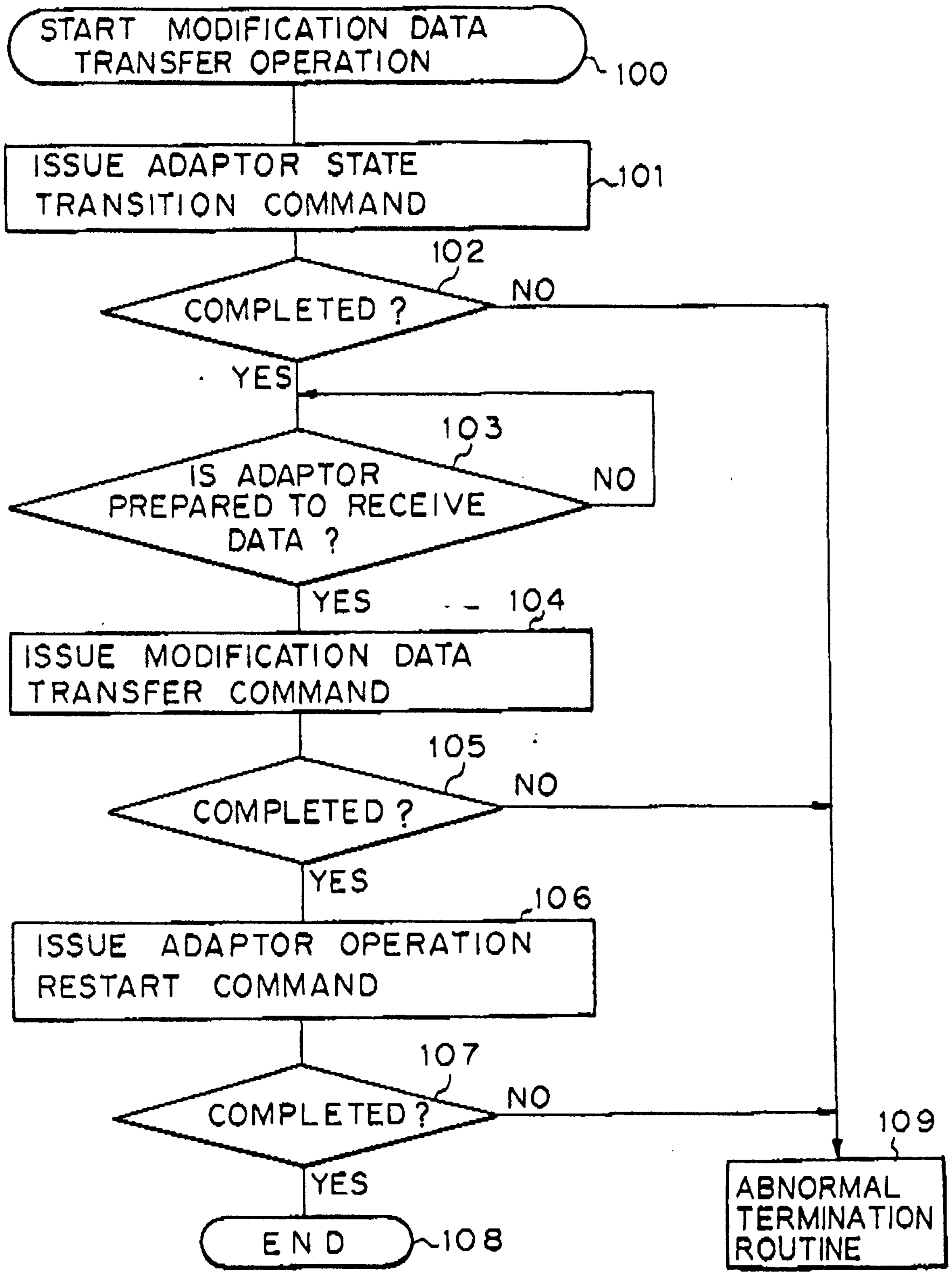

START MODIFICATION DATA TRANSFER OPERATION
100
ISSUE ADAPTOR STATE TRANSITION COMMAND
101
102
COMPLETED ?
NO
YES
103
IS ADAPTOR PREPARED TO RECEIVE DATA ?
NO
YES
104
ISSUE MODIFICATION DATA TRANSFER COMMAND
105
COMPLETED ?
NO
YES
106
ISSUE ADAPTOR OPERATION RESTART COMMAND
107
COMPLETED ?
NO
YES
END
108
109
ABNORMAL TERMINATION ROUTINE

Fig. 6

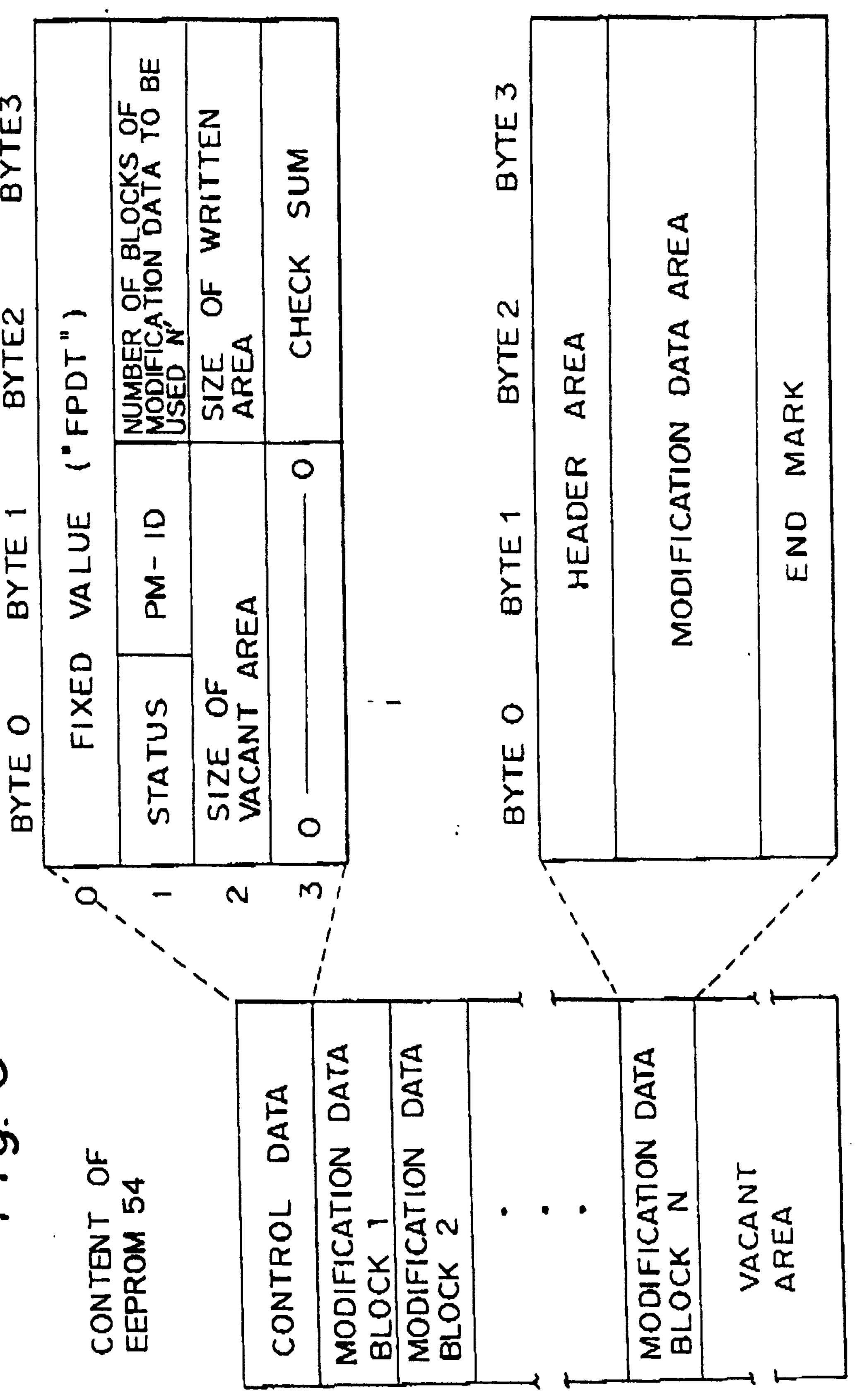

CONTENT OF EEPROM 54
CONTROL DATA
MODIFICATION DATA BLOCK 1
MODIFICATION DATA BLOCK 2
MODIFICATION DATA BLOCK N
VACANT AREA
BYTE 0
BYTE 1
BYTE2
BYTE3
0
1
2
3
FIXED VALUE ("FPDT")
STATUS
PM- ID
NUMBER OF BLOCKS OF MODIFICATION DATA TO BE USED N'
SIZE OF VACANT AREA
SIZE OF WRITTEN AREA
0 ——— 0
CHECK SUM
BYTE 0
BYTE 1
BYTE 2
BYTE 3
HEADER AREA
MODIFICATION DATA AREA
END MARK

Fig. 7

CONTENT OF HEADER

| | BYTE 0 | BYTE 1 | BYTE 2 | BYTE 3 |
|---|---|---|---|---|
| 0 | FIXED VALUE ("DBLK") | | | |
| 1 | CURRENT VERSION TO BE MODIFIED | | NEW VERSION | |
| 2 | SIZE OF BLOCK | | | |
| 3 | 0 ——————— 0 | | CHECK SUN | |

Fig. 8A

CONTENT

| | BYTE 0 | BYTE 1 | BYTE 2 | BYTE 3 |
|---|---|---|---|---|
| 0 | TYPE (00) | ADDRESS | | |
| 1 | OLD DATA | | NEW DATA | |

Fig. 8B

| | | | | |
|---|---|---|---|---|
| 0 | TYPE (01) | ADDRESS | | |
| 1 | NEW DATA 0 | | NEW DATA 1 | |

Fig. 9

| BYTE 0 | BYTE 1 | BYTE 2 | BYTE 3 |
|---|---|---|---|
| FIXED VALUE "F F F F F F F F" | | | |

Fig. 10A · $\overline{I/O}$ COMMAND

Fig. 10B · AOPD

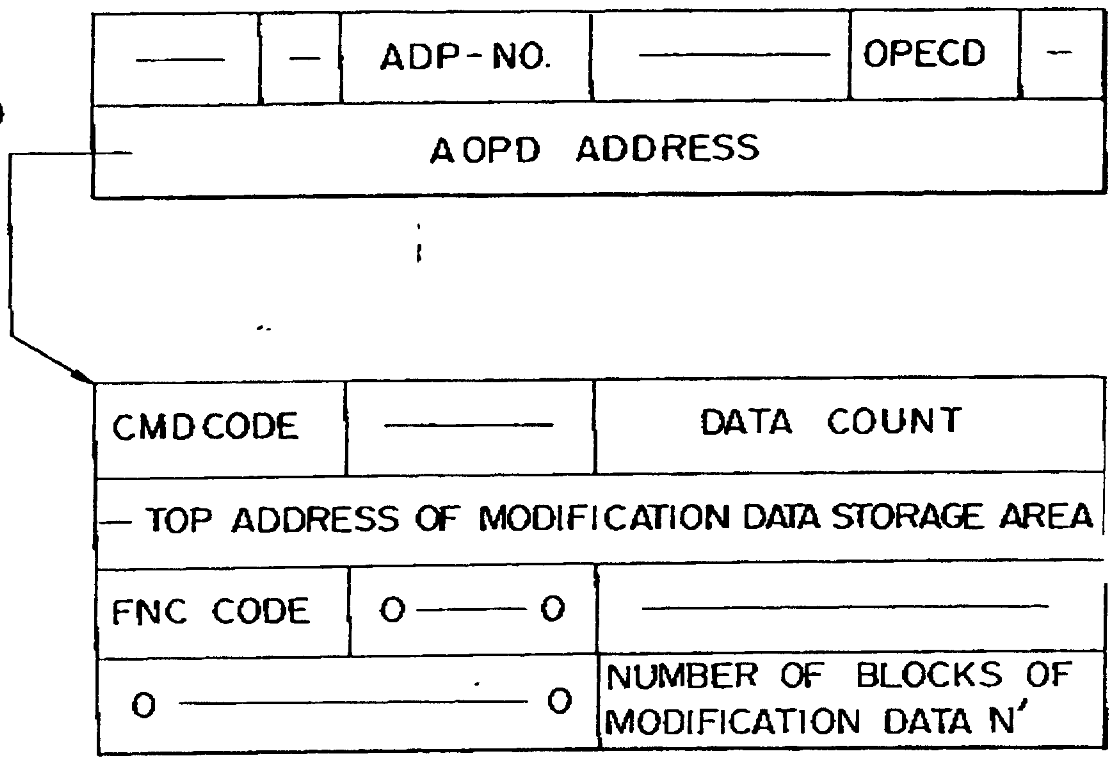

ADP-NO.
OPECD
AOPD ADDRESS
CMDCODE
DATA COUNT
TOP ADDRESS OF MODIFICATION DATA STORAGE AREA
FNC CODE
0 — 0
0 — 0
NUMBER OF BLOCKS OF MODIFICATION DATA N'

Fig. 11

| FNC CODE | OPERATION |
|---|---|
| 0 0 | NEW OPEN |
| 0 1 | APPEND OPEN |
| 0 2 | PUT |
| 0 3 | CLOSE |
| 0 4 | CLEAR |
| 0 5 | VERIFY |

Fig. 12

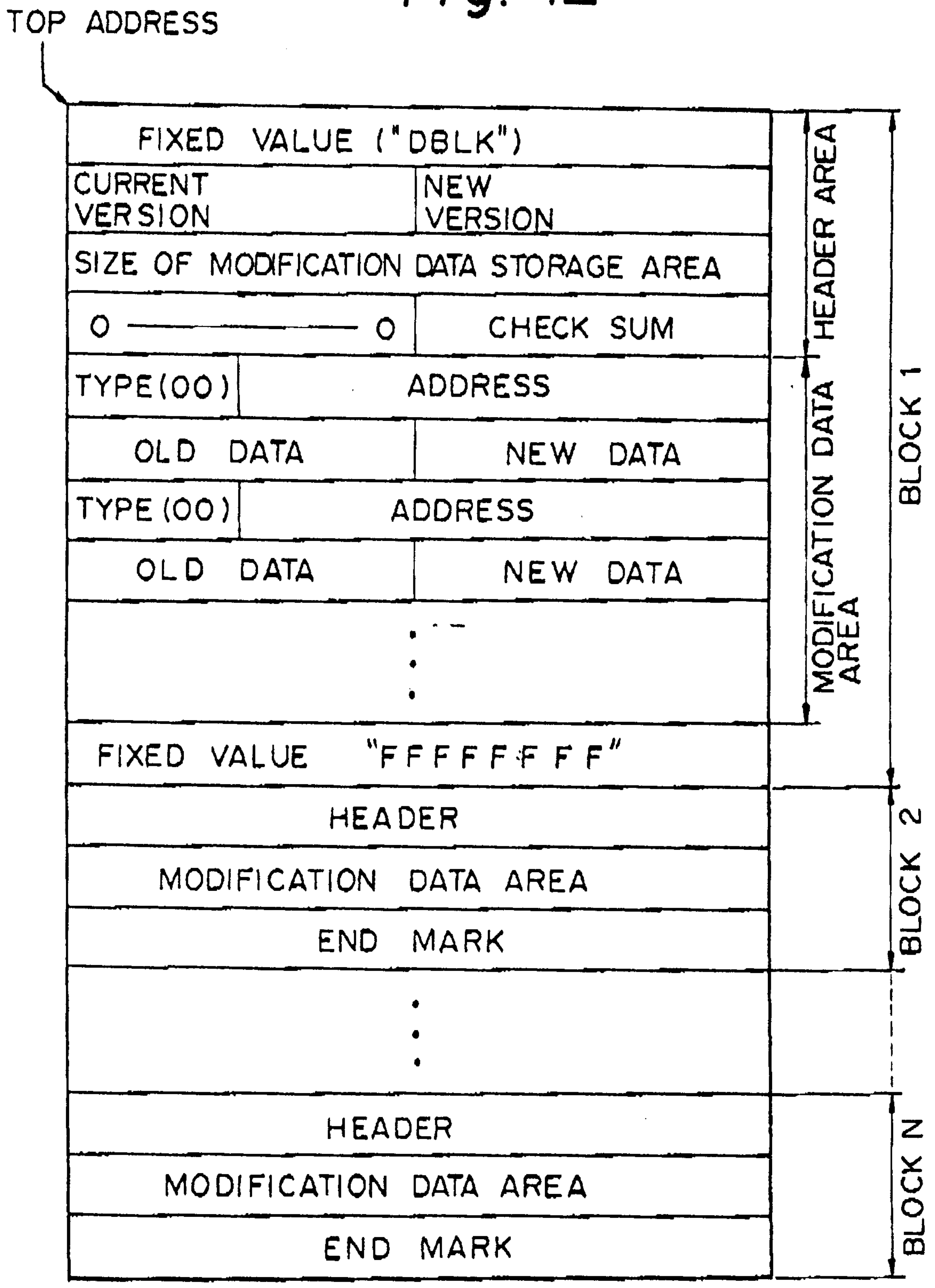

TOP ADDRESS
FIXED VALUE ("DBLK")
CURRENT VERSION
NEW VERSION
SIZE OF MODIFICATION DATA STORAGE AREA
0 —— 0
CHECK SUM
TYPE(00)
ADDRESS
OLD DATA
NEW DATA
TYPE(00)
ADDRESS
OLD DATA
NEW DATA
FIXED VALUE "FFFFFFFF"
HEADER
MODIFICATION DATA AREA
END MARK
HEADER
MODIFICATION DATA AREA
END MARK
HEADER AREA
MODIFICATION DATA AREA
BLOCK 1
BLOCK 2
BLOCK N

Fig. 13A

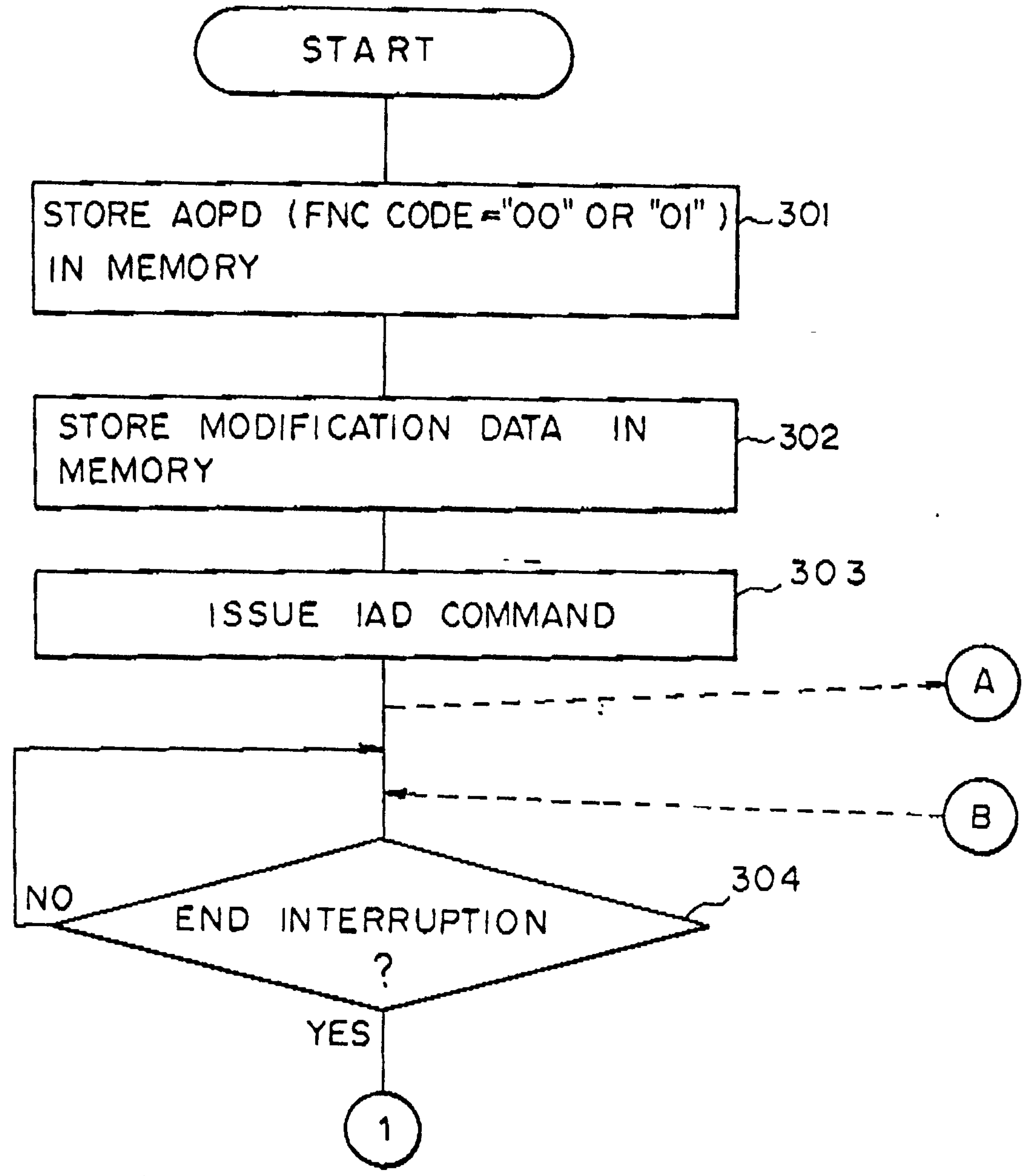

START
STORE AOPD (FNC CODE="00" OR "01") IN MEMORY
301
STORE MODIFICATION DATA IN MEMORY
302
ISSUE IAD COMMAND
303
A
B
END INTERRUPTION ?
304
NO
YES
1

Fig. 13B

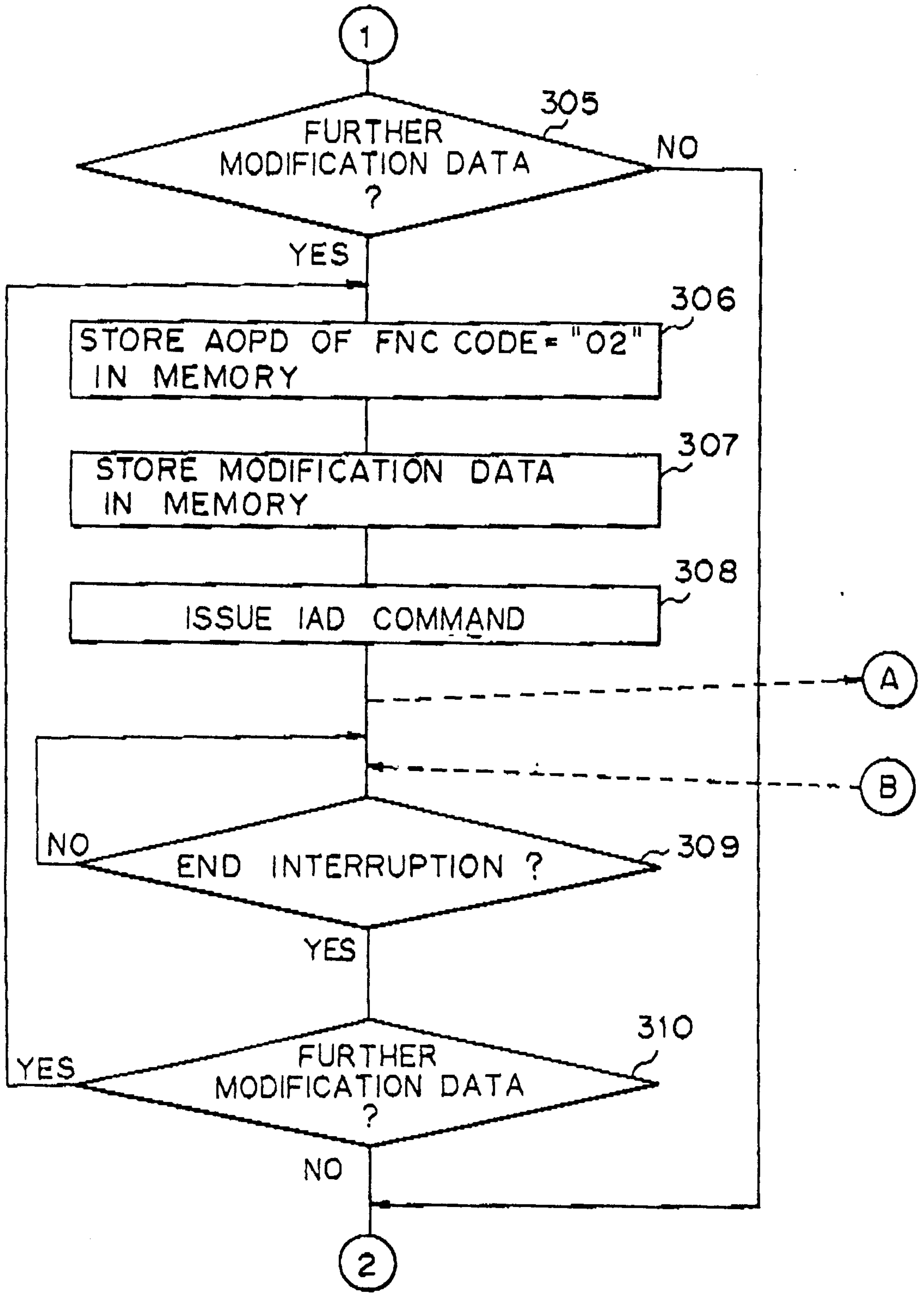

1
FURTHER MODIFICATION DATA ?
305
NO
YES
STORE AOPD OF FNC CODE = "02" IN MEMORY
306
STORE MODIFICATION DATA IN MEMORY
307
ISSUE IAD COMMAND
308
A
B
NO
END INTERRUPTION ?
309
YES
FURTHER MODIFICATION DATA ?
310
YES
NO
2

Fig. 13C

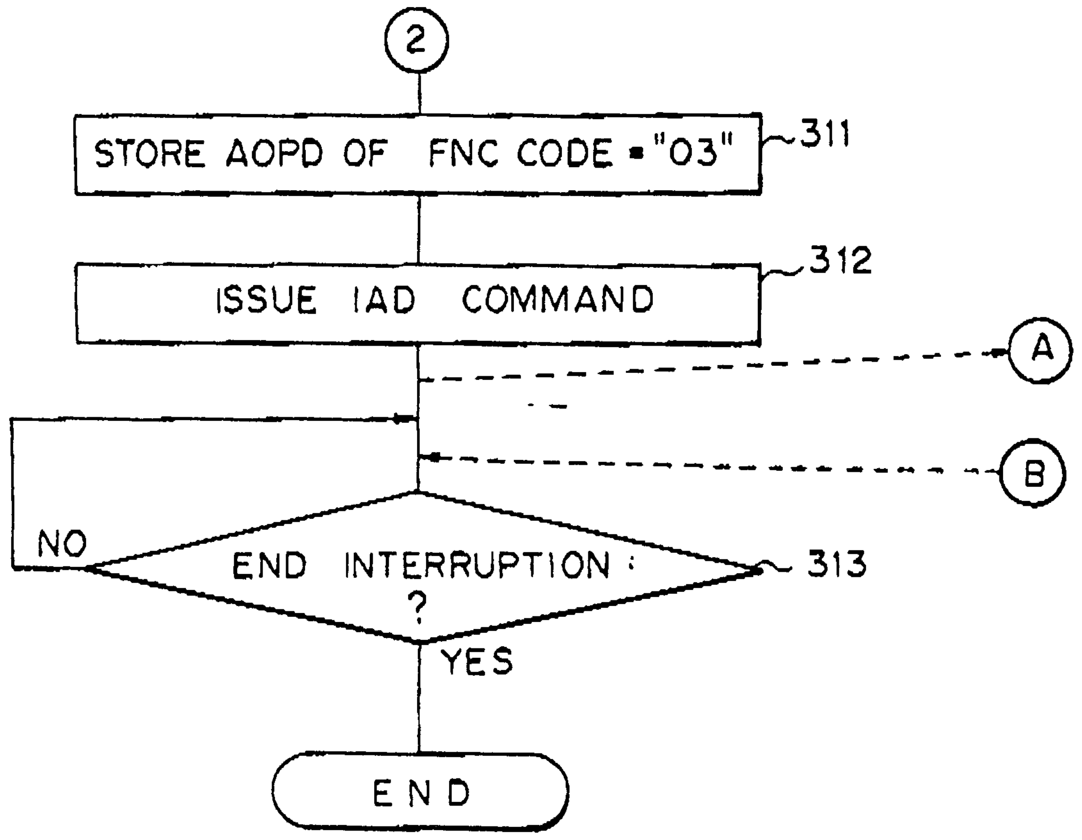

2
STORE AOPD OF FNC CODE = "03"
311
ISSUE IAD COMMAND
312
A
B
NO
END INTERRUPTION ?
313
YES
END

Fig. 14A

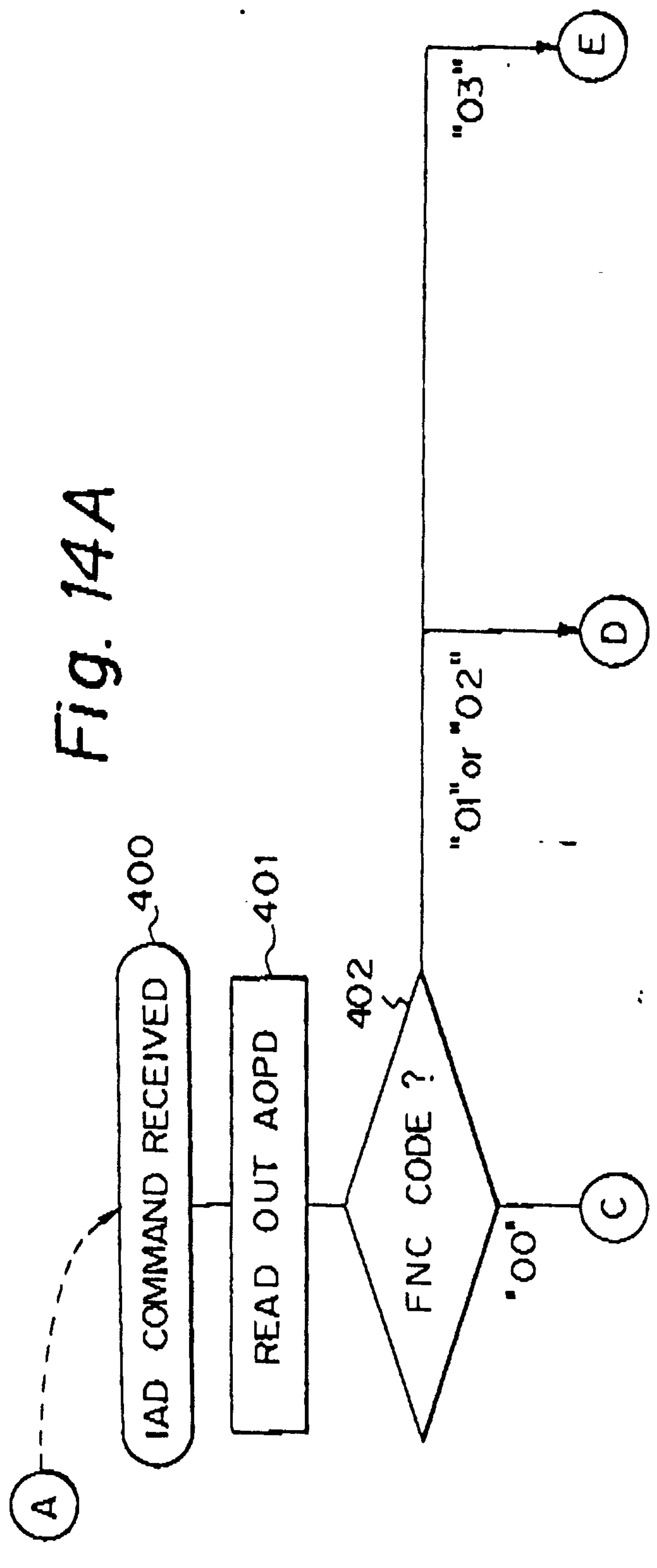


Fig. 14D

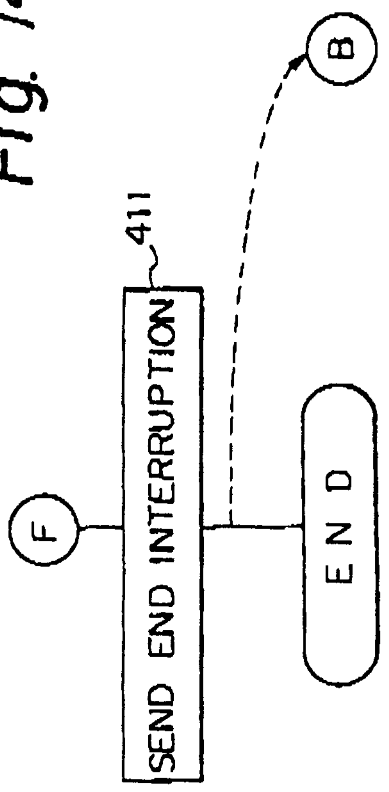


Fig. 14C

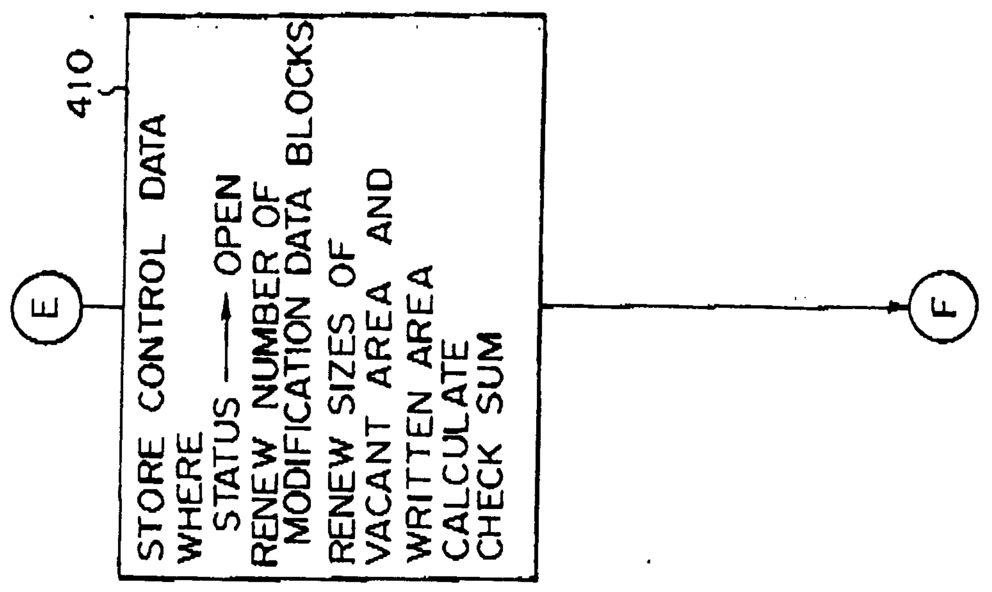

E
410
STORE CONTROL DATA WHERE
STATUS ——→ OPEN
RENEW NUMBER OF MODIFICATION DATA BLOCKS
RENEW SIZES OF VACANT AREA AND WRITTEN AREA
CALCULATE CHECK SUM
F

Fig. 14B

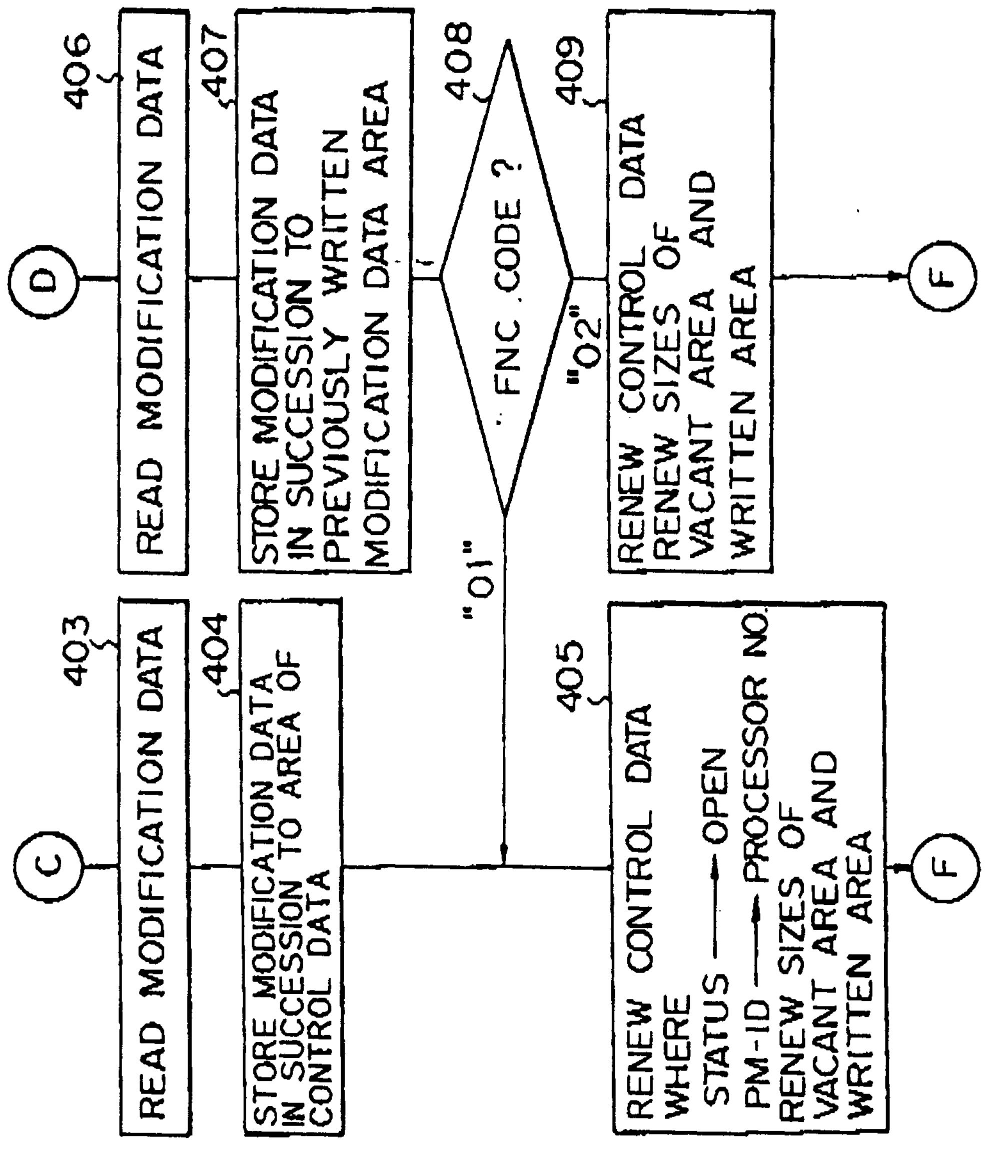

C
403
READ MODIFICATION DATA
404
STORE MODIFICATION DATA IN SUCCESSION TO AREA OF CONTROL DATA
405
RENEW CONTROL DATA WHERE
STATUS ——→ OPEN
PM-ID ——→ PROCESSOR NO.
RENEW SIZES OF VACANT AREA AND WRITTEN AREA
F
D
406
READ MODIFICATION DATA
407
STORE MODIFICATION DATA IN SUCCESSION TO PREVIOUSLY WRITTEN MODIFICATION DATA AREA
408
FNC CODE ?
"01"
"02"
409
RENEW CONTROL DATA
RENEW SIZES OF VACANT AREA AND WRITTEN AREA
F

Fig. 15

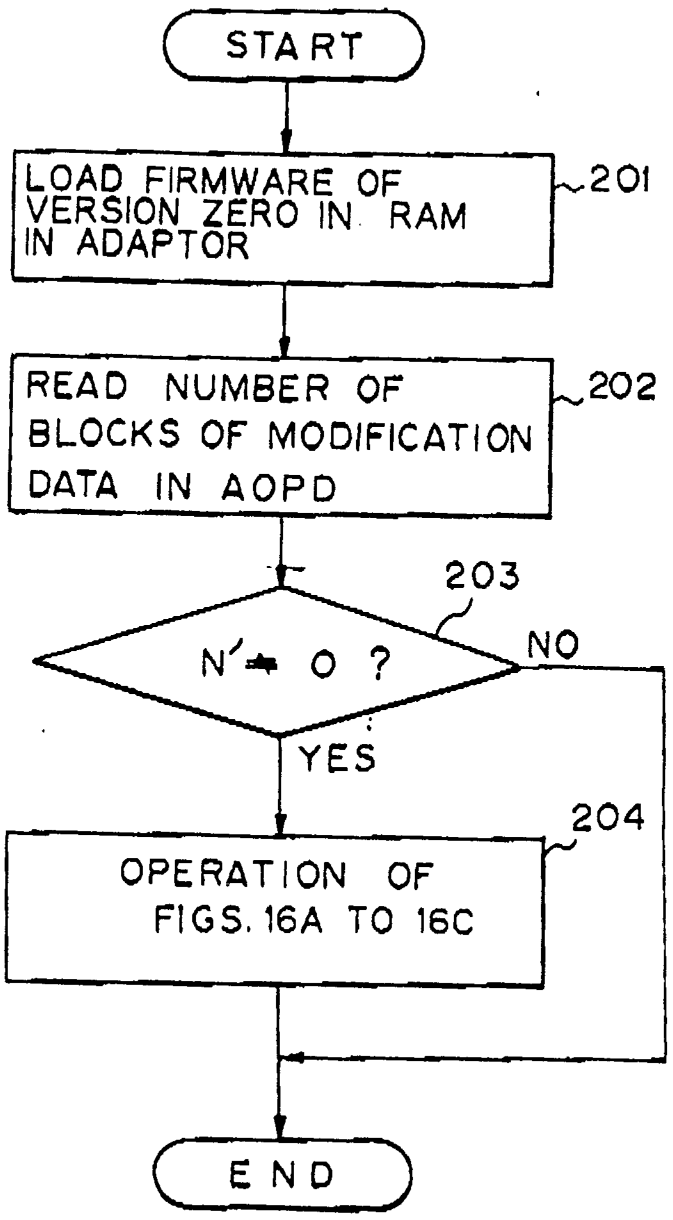

START
LOAD FIRMWARE OF VERSION ZERO IN RAM IN ADAPTOR
201
READ NUMBER OF BLOCKS OF MODIFICATION DATA IN AOPD
202
203
N′ ≠ 0 ?
NO
YES
204
OPERATION OF FIGS. 16A TO 16C
END

Fig. 16A

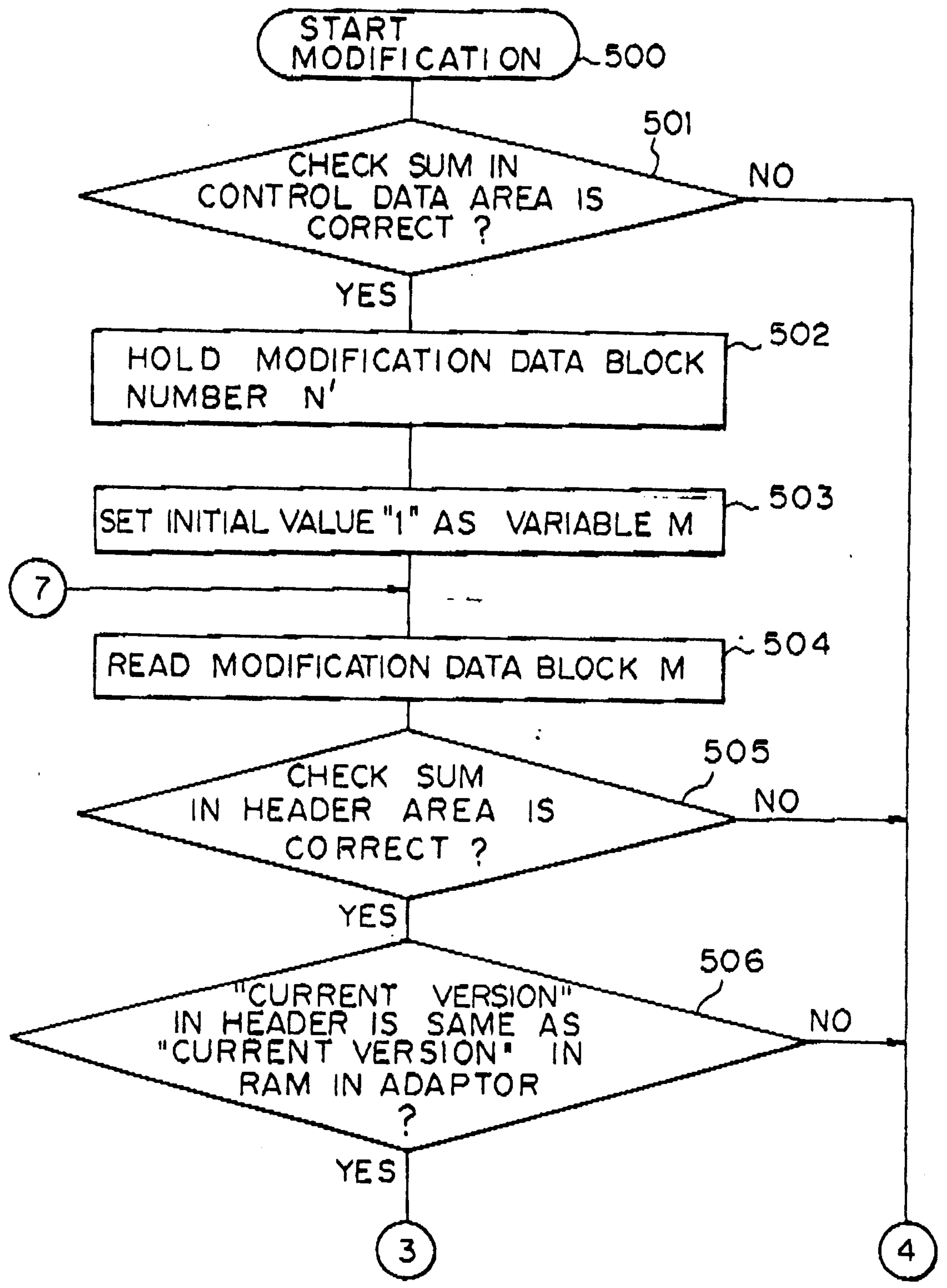

START MODIFICATION
500
501
CHECK SUM IN CONTROL DATA AREA IS CORRECT ?
NO
YES
HOLD MODIFICATION DATA BLOCK NUMBER N'
502
SET INITIAL VALUE "1" AS VARIABLE M
503
7
READ MODIFICATION DATA BLOCK M
504
505
CHECK SUM IN HEADER AREA IS CORRECT ?
NO
YES
506
"CURRENT VERSION" IN HEADER IS SAME AS "CURRENT VERSION" IN RAM IN ADAPTOR ?
NO
YES
3
4

Fig. 16B

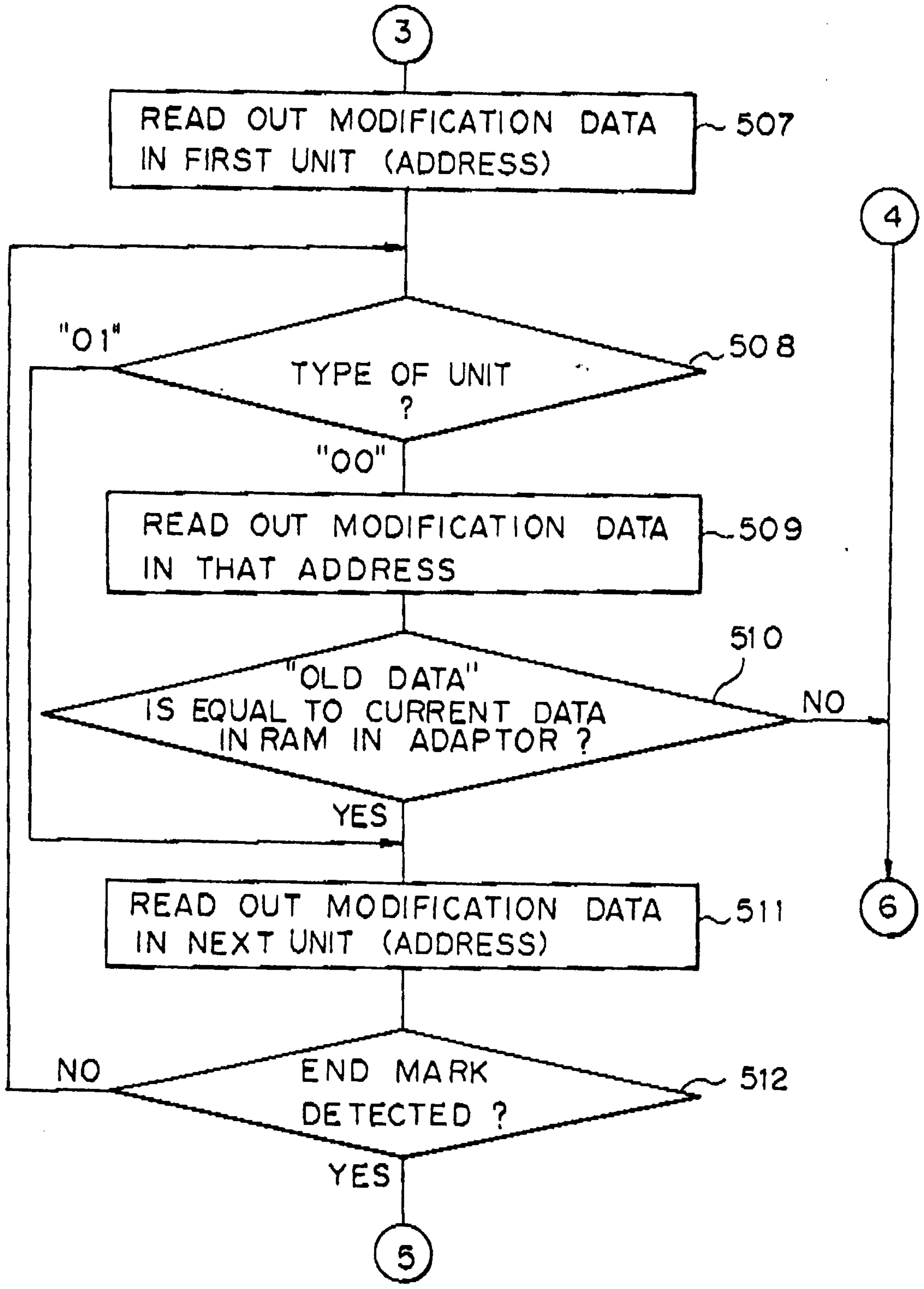
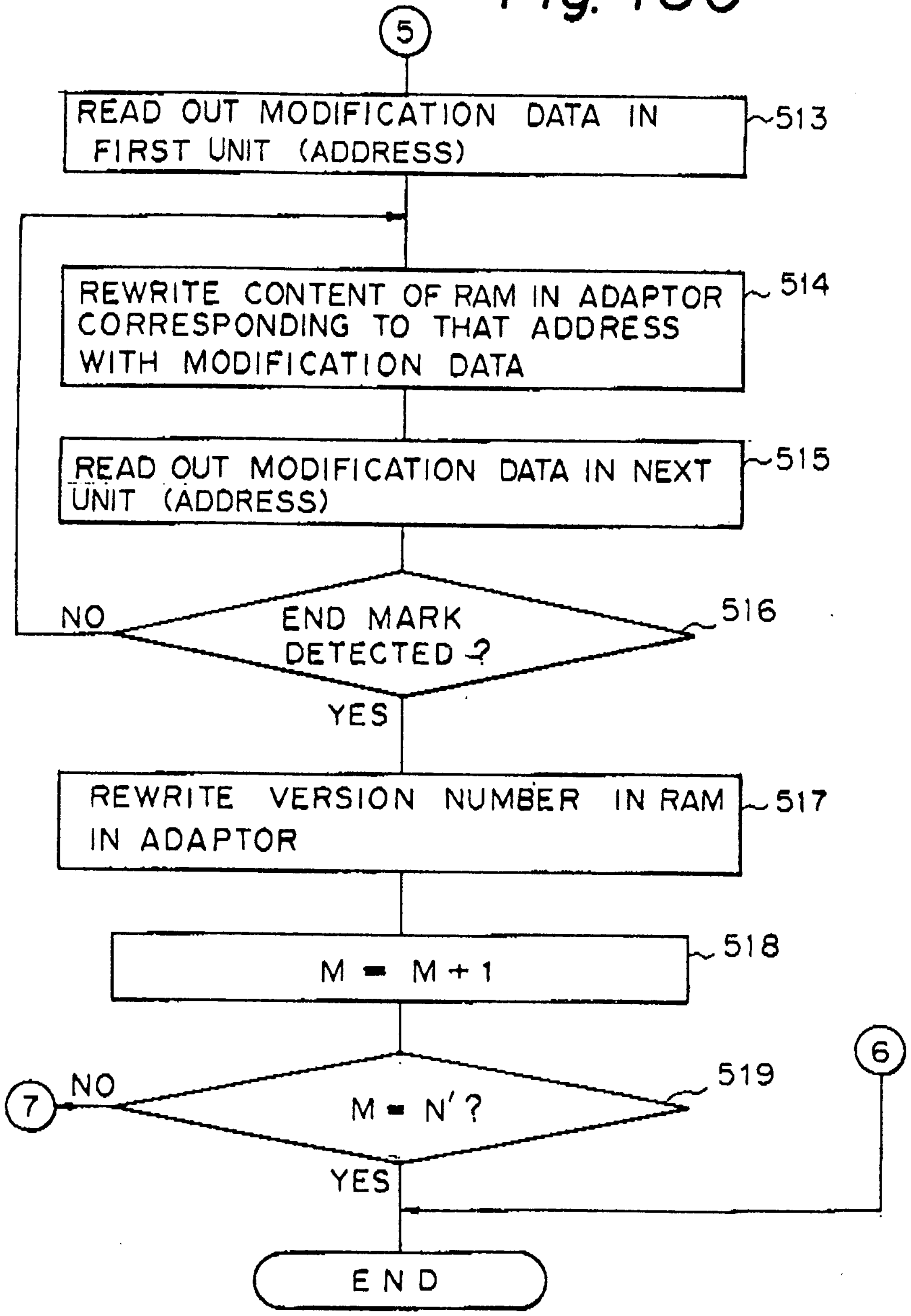

3
READ OUT MODIFICATION DATA IN FIRST UNIT (ADDRESS)
507
4
"01"
TYPE OF UNIT ?
508
"00"
READ OUT MODIFICATION DATA IN THAT ADDRESS
509
510
"OLD DATA" IS EQUAL TO CURRENT DATA IN RAM IN ADAPTOR ?
NO
YES
READ OUT MODIFICATION DATA IN NEXT UNIT (ADDRESS)
511
6
NO
END MARK DETECTED ?
512
YES
5

Fig. 16C


5
READ OUT MODIFICATION DATA IN FIRST UNIT (ADDRESS)
513
REWRITE CONTENT OF RAM IN ADAPTOR CORRESPONDING TO THAT ADDRESS WITH MODIFICATION DATA
514
READ OUT MODIFICATION DATA IN NEXT UNIT (ADDRESS)
515
END MARK DETECTED ?
516
NO
YES
REWRITE VERSION NUMBER IN RAM IN ADAPTOR
517
M = M + 1
518
M = N' ?
519
NO
7
YES
6
END